Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

Módulo de visión artificial del robot humanoide HOAP3. Aplicación al seguimiento de objetivos móviles.

Autor: Alberto Peña Baeza

Tutor: Paolo Pierro

Director: Daniel Hernández García

Leganés, Septiembre 2010

Título: Módulo de visión artificial del robot humanoide HOAP3. Aplicación al seguimiento de objetivos móviles.

Autor: Alberto Peña Baeza

Director: Paolo Pierro y Daniel Hernández García.

EL TRIBUNAL

Presidente: Santiago Martínez de la Casa Díaz.

Vocal: Teresa Leguey Galán.

Secretario: Miguel González-Fierro Palacios.

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 5 de Octubre de 2010 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

"When a distinguished but elderly scientist states that something is possible, he is almost certainly right.

When he states that something is impossible, he is very probably wrong."

Arthur C. Clarke (1917-2008)

A mi padre.

“A veces el hombre más pobre deja a sus hijos la herencia más rica.”

Ruth E. Renkel

AGRADECIMIENTOS

Es difícil redactar estas líneas sin regresar al primer día de clase en la universidad. Aquellos primeros días donde empezábamos a conocernos, sin saber lo que nos esperaba en estos años que hemos permanecido juntos. Por ello quiero agradecerles a todos mis amigos de la universidad los buenos momentos vividos y apoyo en los malos. Demostrándonos unos a otros que no hay nada imposible.

A mi tutor Paolo que siempre ha estado dispuesto a responder todas las dudas que tenía. Un agradecimiento especial a Daniel y Miguel por la enorme paciencia que han tenido y la cantidad de horas que han estado conmigo al pie del cañón pese a tener cosas más importantes que hacer.

A mis amigos de toda la vida por seguir aguantándome. Y a toda la gente que me ha apoyado y no puedo nombrar individualmente.

Una especial mención a David López por su ayuda y muestra de interés en mi trabajo.

A Paula por estar siempre cuando la necesito y apoyar mis ideas por absurdas que sean.

A mi madre y mi hermano por el apoyo que me han dado siempre en los buenos y malos momentos. A mi padre por inculcarme el interés por la ingeniería.

A todos vosotros GRACIAS.

RESUMEN

Llevamos muchos años siendo conscientes de la importancia de la robótica en al ámbito industrial. Sin embargo la sociedad todavía no es consciente de la importancia que está tomando la robótica en el ámbito doméstico, ya sea mediante humanoides u otros robots de asistencia.

El objeto de este proyecto es conseguir que el robot humanoide HOAP3 detecte e identifique determinados objetos y sea capaz de seguir su trayectoria, obteniendo la posición relativa del objeto con respecto el robot.

Para ello se parte de un código existente desarrollado en C++. A partir de dicho código se pretende implementar otro más sencillo e intuitivo mediante la utilización de las librerías OpenCV. Estas librerías presentan numerosas ventajas ya que se trata de código abierto, optimizado y desarrollado para aplicaciones en tiempo real, independiente de sistema operativo o hardware, etcétera.

Para implementar el código con las librerías OpenCV se han estudiado e implementado los distintos algoritmos de segmentación de imágenes, detección de contornos. En este proyecto se ha trabajado con los distintos espacios de color para buscar las soluciones más robustas.

También se ha procedido al estudio de las características principales de la visión estéreo para optimizar el resultado obtenido de aplicar el algoritmo de cálculo de distancias con respecto al robot.

Palabras clave: segmentar, visión estéreo, funcionalidad, asistencia.

ABSTRACT

It has been while since we are aware of the importance of robotics in industrial environments. However society is not concern yet about the importance it is getting in domestic environments, even by humanoids or other assistance robots.

The purpose of this thesis is to make the HOAP3 humanoid be able to detect and identify certain objects as well as follow their trajectory, getting its position respect the robot.

Initially we have an existing code implemented in C++. From this code it is expected to develop another one more simple and intuitive using the OpenCV libraries. These libraries have several advantages, it is open source, optimized and developed for real time applications, independent of the Operative System or hardware, etcetera.

To develop the code with the OpenCV libraries many algorithms such as image segmentation and contour detection have been studied. In this project we have been working with different color models to obtain the better results possible.

To optimize the functionality of the code it is been necessary to study the main characteristics of stereo vision.

Keywords: stereo vision, segmentation, functionality, assistance.

INDICE

INDICE DE FIGURAS

1.INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

Vivimos en una sociedad rodeada de tecnología, la cual evoluciona cada vez más rápido alcanzando unos límites que hace no mucho parecían insospechados. Este desarrollo tan vertiginoso nos permite llevar una vida más confortable. Una disciplina que está en auge y sobre la que se basa este proyecto es la robótica, a la cual se le intenta dar cada vez más una aplicación doméstica.

En los últimos años se ha dado un fuerte impulso a la investigación en robots humanoides, desarrollándose robots con alta capacidad de manipulación, interacción y movilidad. Algunos ejemplos son el WABIAN-2 de la Universidad de Waseda (Ogura et al., 2006), el ASIMO de Honda (Sakagami et al., 2002), el HRP-2 del National Institute of Advanced Industrial Science and Technology of Japan (Kaneko et al., 2004) o el RH-1 diseñado en la Universidad Carlos III de Madrid (Arbulú, Kaynov, Cabas, y Balaguer, 2009). Pero para que estos robots interaccionen con el ser humano y se introduzcan plenamente en su entorno, es necesario aumentar su seguridad, manipulabilidad, estabilidad y su capacidad de interacción.

Las aplicaciones robóticas futuras prevén un mundo en que los robots y los humanos interactúan naturalmente con los humanos en el hogar, la oficina, centros de entretenimiento, etcétera, ayudando en la realización de actividades comunes y mejorando la calidad de vida. Aunque continuamente ocurren grandes avances en el campo de la robótica, aún estamos lejos de conseguir un robot capaz de tal destreza y autonomía. Para llegar a este nivel de adaptación de los robots al mundo humano se pueden considerar dos grandes temas, la capacidad de los robots para desplazarse e interactuar con el entorno, y la capacidad del robot para percibir y comprender ese entorno. En los humanos uno de los sistemas de percepción más importante es la visión. En este trabajo se desarrolló un sistema de visión por computador, basado en las librerías OpenCV, que permite a un robot, HOAP-3, detectar e identificar una serie de objetos en su entorno.

La visión por computador es campo de investigación de un gran interés. La fabricación de cámaras y equipos de procesamiento más baratos y más avanzados y el desarrollo y distribución de mejores herramientas y algoritmos de visión han permitido el crecimiento de este campo. Las librerías de OpenCV (Open Source Computer Vision Library) son unas de estas herramientas que ha jugado un papel esencial en el crecimiento de la visión por computador permitiendo realizar trabajos en visión de forma más productiva.

OpenCV ayuda a estudiantes y profesionales a implementar de manera rápida y eficiente de proyectos de investigación en visión. OpenCV proporciona una infraestructura de de algoritmos maduros de visión por ordenador y machine learning del cual partir en el desarrollo de nuevas implementaciones de soluciones basadas en visión por computador. (Gary Bradski and Adrian Kaehler, 2008)

En este trabajo se desarrollo un algoritmo capaz de detectar objetos, calcular su posición, e identificar, mediante su color y su forma, el objeto detectado. Este sistema de visión se desarrollo para proveer de visión por computador al robot humanoide HOAP-3.

1.2 OBJETIVOS

El principal objetivo de este proyecto final de carrera es el de dotar al robot humanoide HOAP-3 de un sistema de visión artificial que le permita detectar y seguir determinados objetos en su entorno. Además se busca obtener distintos subobjetivos:

- Desarrollar el código de la forma más sencilla posible.
- Calcular la posición del objeto en referencia con el robot.
- Reconocer e identificar diferentes objetos.
- Presentar al usuario del robot la imagen procesada del robot presentando información de los objetos detectados.

1.3 FASES DE DESARROLLO

Para el desarrollo del proyecto se han llevado a cabo las siguientes tareas:

- Toma de confianza con las librerías OpenCV.
  Búsqueda de documentación y estudio de las librerías OpenCV.
- Estudio del estado del arte.
  Lectura y estudio de artículos sobre el estado actual de la visión, detección y tracking de objetos.
- Estudio del código existente.
  Entender el funcionamiento del código proporcionado.
- Desarrollo de un nuevo código.
  Desarrollo de un nuevo código basado en las librerías OpenCV.

1.4 MEDIOS EMPLEADOS

Para el desarrollo del presente proyecto se han empleado los siguientes elementos:

- Un ordenador portátil personal con webcam integrada, en el que ha sido necesaria la instalación del sistema operativo Linux en su última versión de Ubuntu: 10.04 Lucid Lynx.
- El robot humanoide HOAP-3.

1.5 ESTRUCTURA DE LA MEMORIA

Capítulo 1: Introducción y objetivos.

En este capítulo se expone al lector el propósito de este proyecto así como se describen el estado de desarrollo de la robótica en este momento.

Capítulo 2: Estado del arte.

El segundo capítulo del proyecto versa tanto sobre el estado actual de las tecnologías empleadas como de su evolución hasta este punto.

Capítulo 3: Herramientas y plataforma.

Aquí se introduce al lector en las librerías OpenCV como también se le explica un poco el diseño y morfología del humanoide empleado en el desarrollo del proyecto.

Capítulo 4: Implementación algoritmos.

En este capítulo se describirá más en detalle los medios y metodologías empleadas para dotar al HOAP-3 de la capacidad de detectar objetos y calcular distancias.

Capítulo 5: Resultados, problemática y limitaciones.

Aquí se exponen los principales problemas surgidos durante el desarrollo y prueba del código, así como los resultados obtenidos.

Capítulo 6: Conclusiones y líneas de trabajo futuro.

En el último capítulo se describen brevemente las posibles líneas de trabajo futuro a seguir y se hace una valoración final de los resultados obtenidos.

2.ESTADO DEL ARTE

2.1 VISION POR COMPUTADOR

La visión por computador ([1],[7],[8],[9],[11],[13],[15],[16],[17],[18],[19],[20],[21]) tiene como objetivo la interpretación de escenas a partir de las imágenes suministradas por cámaras, utilizando para ello la potencia de procesamiento de los computadores digitales. En este apartado se describirá el estado en el que se encuentra actualmente la visión por computador, así como sus fundamentos.

La visión por computador, también conocida como visión artificial, es una disciplina que persigue la deducción automática de la estructura y propiedades de un escenario. Dicho escenario posiblemente cambiante será estudiado a partir de imágenes captadas del mismo, estas imágenes pueden ser en blanco y negro o color, tomadas por una o varias cámaras. Actualmente se puede hablar incluso de imágenes procedentes de sensores de naturaleza no visual, como por ejemplo acústicos, térmicos, táctiles.

La visión por computador es una disciplina que engloba todos los procesos que proporcionan de alguna forma del sentido de la vista a una máquina. Se podría decir que: *la visión artificial o comprensión de imágenes describe la deducción automática de la estructura y propiedades de un mundo tridimensional, posiblemente dinámico, bien a partir de una o varias imágenes bidimensionales de ese mundo*. [30]

Conviene destacar que hoy por hoy la visión por computador en algunas ocasiones no es la mejor solución a un problema. En muchas ocasiones en las que el problema a enfrentar es tan complejo que la solución humana es lo mejor, por ejemplo aplicaciones de visión para conducción asistida, que como se ha comprobado en numerosas ocasiones el ser humano reacciona mejor que la máquina ya que intervienen numerosos factores que en muchas ocasiones no son sencillos de parametrizar. Sin embargo también se puede dar el caso contrario, *que las soluciones humanas sean inexactas o subjetivas y en ocasiones lentas y presentan una ausencia de rigor así como una pobre precisión* [31]. No obstante, la solución humana es menos estructurada que la solución artificial y muchos problemas de visión por computador requieren un nivel de inteligencia mucho mayor que el que la máquina pueda ofrecer. El sistema de visión humana puede describir automáticamente una textura en detalle, un borde, un color, una representación bidimensional de una tridimensional, ya que puede diferenciar entre imágenes de diferentes personas, firmas, colores, etcétera, puede vigilar ciertas zonas, diagnosticar enfermedades a partir de radiografías, etc. Sin embargo, aunque algunas de estas tareas pueden llevarse a cabo mediante visión artificial, el software o el hardware necesario no consigue los resultados que serían deseables.

A pesar de las limitaciones expuestas, cada día es mayor el número de aplicaciones de visión por computador. Dentro de todas las aplicaciones posibles, podemos encontrar una clasificación en función de si la visión artificial es una herramienta por sí sola o si por el contrario forma parte de un sistema multisensorial, como es el caso del tema que estamos abordando. Entre todas las aplicaciones posibles algunas son:

- Navegación en robótica.
- Biología, geología y meteorología.
- Medicina.
- Reconocimiento y clasificación.
- Inspección y control de calidad.
- Etcétera.

Desde este punto en adelante toda la información sobre el estado actual de la visión por computador está basada en la aplicación a la robótica, que es el objeto de este proyecto.

Las decisiones en tiempo real basadas en sensores visuales es una aplicación muy demandada para robots móviles. El principal problema encontrado en el campo de la robótica es el de responder de forma adecuada a un estímulo visual en tiempo real. Por lo que la información visual adquiere cada día más importancia. Actualmente el mayor avance en visión por computador consiste en la aplicación de redes neuronales para el aprendizaje por parte del robot.

Las redes neuronales son dispositivos inspirados en la funcionalidad de las neuronas biológicas, aplicados al reconocimiento de patrones que las convierten en aptas para modelar y efectuar predicciones en sistemas muy complejas. Su funcionamiento es como un conjunto de técnicas matemáticas para moldear la conexiones / relaciones entre un conjunto de datos. Surgieron del movimiento conexionista, que nació junto con la Inteligencia Artificial, IA de ahora en adelante, simbólica o tradicional. La IA simbólica se basa en que todo conocimiento se puede representar mediante combinaciones de símbolos, derivadas de otras combinaciones que representan verdades incuestionables o axiomas. Así pues, la IA tradicional asume que el conocimiento es independiente de la estructura que maneje los símbolos, siempre y cuando la "máquina" realice algunas operaciones básicas entre ellos. Un ejemplo de red neuronal es el Perceptrón unicapa ([23],[24]), que es un conjunto de neuronas conectadas entre sí, las cuales producirán una salida individual. En la Figura 1 se puede observar un sistema formado por Perceptrones unicapa que reciben distintas señales, generando una salida cada uno.

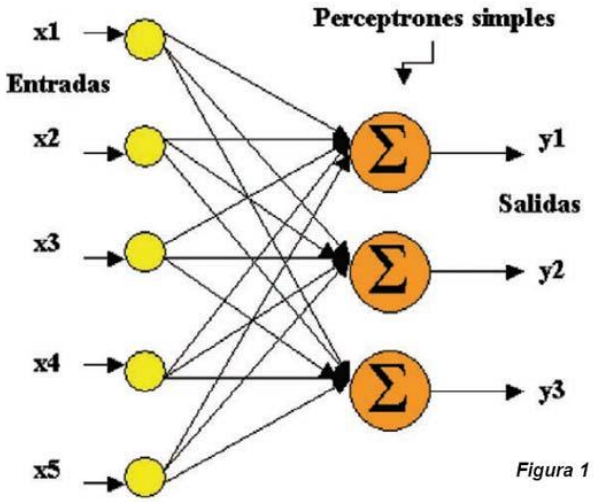


Figura 1: conjunto de Perceptrones unicapa

Existen tres métodos de aprendizaje para un Perceptrón:

- Aprendizaje supervisado: se presentan al Perceptrón unas entradas con las correspondientes salidas que se quiere que sean aprendidas.
- Aprendizaje no supervisado: solo se presentan al Perceptrón las entradas y, para esas entradas, la red debe dar una salida parecida.
- Aprendizaje por refuerzo: se combinan las dos anteriores, y cada cierto tiempo se presenta a la red una valoración global de cómo lo está haciendo.

Se podría pensar que el Perceptrón tiene una capacidad ilimitada de aprendizaje, sin embargo Marvin Minsky y Seymour Papert en su libro "Perceptrons" (MIT press, 1969) establecen que el Preceptrón unicapa es incapaz de aprender las funciones que no fuesen linealmente separables. Pese a ello las redes neuronales parece que darán mucho de qué hablar en robótica.

2.2 DETECCION DE OBJETOS

La capacidad de un robot para interactuar con el medio viene dada por la posibilidad de extraer información de su entorno. Dicha extracción de información puede ser en forma de detección de objetos presentes en el entorno, por lo que es una capacidad muy útil de la visión por computador en el campo de la robótica. Esto confiere a los robots autónomos o de teledirección asistida un mayor poder de decisión independientemente de si son del tipo humanoide, rover, industriales, etcétera.

Con la evolución en robótica que se está viviendo en estos momentos no cabe pensar en un robot autónomo sin la capacidad de distinguir los objetos del entorno que lo rodea.

Actualmente se pueden distinguir dos metodologías principales para la detección de objetos:

- Basada en la transformación de la imagen mediante segmentaciones.
- Haar Feature-based Cascade Classifier.

La primera de las metodologías consiste en realizar transformaciones a la imagen para ir sustrayendo información que resulte útil para su análisis ([10],[12],[27]). En [22] se puede encontrar una completa guía con todas las operaciones que podemos aplicar a una imagen. Una de las operaciones más importantes es la segmentación, dentro de la cual tenemos varias técnicas.

Segmentación basada en umbralización

El histograma de una imagen permite ver la frecuencia relativa de aparición de cada nivel de luminancia de una imagen. El proceso de umbralización basada en el histograma consiste en agrupar los píxeles según sus niveles de intensidad luminosa. En la Figura 2 puede observarse el histograma de cada uno de los canales de la imagen.

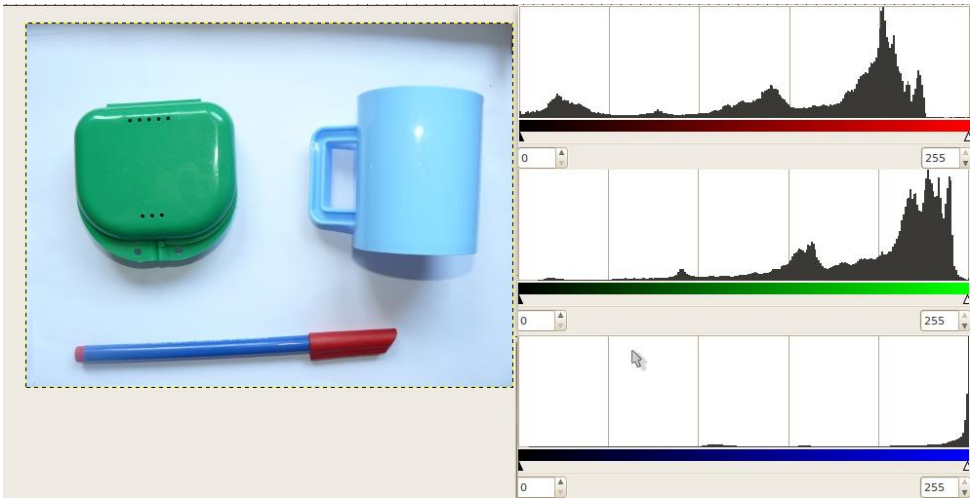


Figura 2: Histograma de los 3 canales RGB de una imagen

Hay que tener en cuenta que dependiendo de los objetos que tenga en la imagen el histograma podrá tener varios montículos en un mismo canal. Por lo que interesará separar los objetos del fondo. El problema surge cuando las condiciones de iluminación, fondo u objetos, no son homogéneas. En ese caso habrá que desarrollar un método de cálculo de los umbrales de forma automática:

- Método P-cuantil: requiere información del área del objeto a detectar. Útil para reconocimiento de caracteres.

- Búsqueda de mínimos: obteniendo los puntos mínimos de los valles formados por los montículos y asignándolos como valores iniciales de los umbrales buscados. Véase Figura 3.

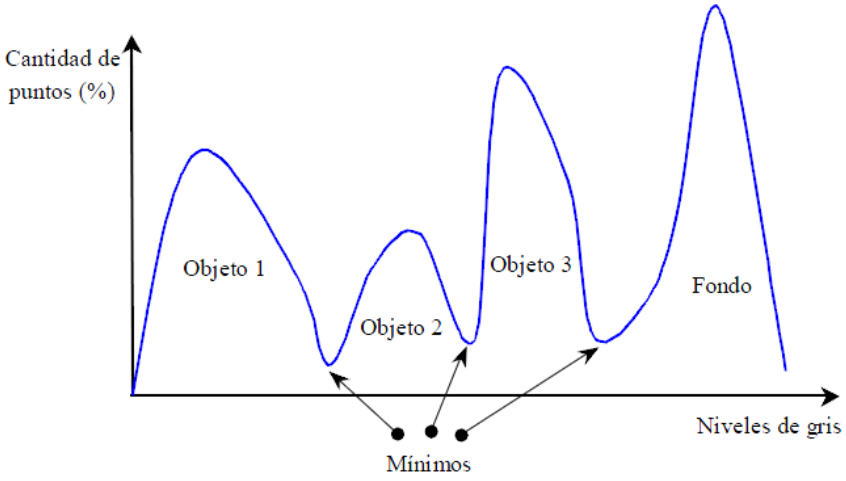


Figura 3: Mínimos del histograma

- Método de Otsu: define el mejor umbral posible. El mejor umbral es aquél cuya suma de pesos de cada grupo de varianzas se minimiza, siendo los pesos las probabilidades respectivas de cada grupo.

Segmentación basada en bordes

Se trata del primer grupo de métodos de segmentación utilizado. Se basa en los bordes encontrados en la imagen por los detectores. La diferencia entre unos métodos y otros radica en la estrategia para la construcción, así como en la cantidad de información previa que se incorpora en el método. Algunos de los métodos son:

- Relajación de bordes: este método consiste en determinar mediante la vecindad la pertenencia de un píxel a un borde, así incrementamos la calidad de los resultados. Un borde débil entre dos bordes fuertes en su vecindad, es altamente probable que el borde débil intermedio sea parte de un contorno. Un borde fuerte aislado sin refuerzo en su vecindad probablemente no sea parte de un contorno verdadero.
- Extracción de la frontera: la frontera son bordes unidos que caracterizan la forma de un objeto. Útil para calcular rasgos geométricos.
- Transformada de Hough: sirve para vectorizar una imagen de líneas y contornos. Partiendo de una imagen en la cual se han extraído sus contornos se procede a convertir estos en una serie de vectores. Cálculo lento.
- Transformada Radon: proyecta todos los puntos de la imagen en una línea diagonal que va girando de 0 a 180 grados. Para cada

grado se tiene un vector que representa a la recta con esa inclinación y el número de puntos que han incluido en ella.

Segmentación orientada a regiones

Los métodos de segmentación por regiones, a diferencia de los anteriores que buscan bordes se basan en la construcción de las regiones directamente. Cada una de las regiones resultantes de este tipo de segmentación han de ser homogéneas y maximales. Por maximal se entiende que los criterios de homogeneidad no se cumplirán tras la unión de una región con una de sus adyacentes. Los métodos más comunes son:

- Unión de regiones: se considera a cada píxel de la imagen como una región. Obviamente cada región inicial cumple la condición de homogeneidad, pero no necesariamente el criterio de ser región maximal. El algoritmo estaría formado por tres pasos. Primero, definir una segmentación que cumpla el criterio de homogeneidad. Segundo, definir un criterio para unir regiones. Tercero, unir regiones adyacentes que cumplan los criterios de unión y homogeneidad hasta ser maximales.
- Crecimiento de regiones por agregación de píxeles: el crecimiento de regiones se realiza a partir de un conjunto de píxeles semilla. A partir de ellos se hace crecer la región añadiendo a dichos píxeles semilla aquellos píxeles vecinos que tengan características similares, ya sea nivel de gris, textura, etcétera.
- Algoritmos de crecimiento con mezclado: el problema de los métodos anteriores es el caso probable en el cual un píxel pertenezca a varias zonas. Para ello se utiliza la media aritmética m_i y la desviación estándar de una región de n puntos, como se muestra en las (1) y (2).

$$m_i = \frac{1}{n} \cdot \sum_{(k,l)\in R_i} f(k,l) \qquad (1)$$

$$\sigma_i = \sqrt{\frac{1}{n} \cdot \sum_{(k,l)\in R_i} (f(k,l) - m_i)^2} \qquad (2)$$

De tal forma que para decidir si dos regiones R_1 y R_2 deben mezclarse se considera la ecuación:

$$|m_1 - m_2| < k \cdot \sigma_i \quad \text{siendo i=1,2}$$

Segmentación por moción

Se trata de un campo de la segmentación con mucho auge. Se basa en la comparación de varias imágenes tomadas en el tiempo por lo que se puede detectar el objeto en movimiento y sus características. Muy útil para aplicaciones en robótica y control de calidad.

Una vez segmentada la imagen original se procede a un análisis, en dicho análisis pueden ser interesantes características como el área, perímetro, momentos, curvatura, etcétera.

La segunda de las principales metodologías es el Haar Feature-based Cascade Classifier [2],[3],[4],[5],[32]y [25]. Esta metodología consiste en entrenar a mi sistema para que sea capaz de reconocer objetos en cualquier entorno y bajo cualquier circunstancia. Desarrollada por Paul Viola [3] Algunos ejemplos de esta metodología pueden encontrarse en [4][2] y [25]. A continuación se explica más detalladamente su funcionamiento.

Este método se basa en la creación de clasificadores mediante los cuales yo podré entrenar a mi sistema para que detecte los objetos pertenecientes a dicha clase. Conviene definir bien ciertos conceptos de esta metodología antes de profundizar más [6] y [6].

- Boosting: grupo de clasificadores. La decisión acerca de la clasificación se hace mediante una ponderación de las decisiones del grupo de clasificadores.
- Haar classifier: aplicación para la detección de objetos basado en un uso inteligente de boosting.

Para crear un clasificador (namely a *cascade of boosted classifiers working with haar-like features)* basado en el método de Viola-Jones, para cualquier tipo de objeto, es necesario construir dos conjuntos de imágenes independientes.

- Imágenes positivas: conjunto de imágenes que contienen muestras del objeto que deseamos detectar, es decir, aquel para el que diseñamos el clasificador. Se requieren unas 5000 imágenes de este tipo.
- Imágenes negativas: conjunto de imágenes que nunca contendrán el objeto que deseamos detectar. Se necesitan unas 3000 imágenes negativas.

Una vez obtenidos todos los datos necesarios de las imágenes suministradas se procede con el entrenamiento, que tendrá en cuenta características como:

- Localización del clasificador resultante.
- Muestras positivas.
- Dimensiones de las muestras.
- Mínima tasa de acierto en positivas.
- Máxima tasa de error en negativas.
- Etcétera.

Después de que el clasificador ha sido entrenado puede ser aplicado a una ROI, del mismo tamaño que la usada en el entrenamiento, de una imagen de entrada. Si el objeto del clasificador se encuentra en la ROI, éste devolverá un 1 sino un 0. Para buscar en toda la imagen simplemente hay que mover la ROI. El clasificador está definido de tal forma que es fácil modificar su tamaño para poder detectar objetos más o menos grandes, por lo tanto si queremos detectar un objeto de tamaño indefinido habrá que realizar este procedimiento varias veces.

La palabra “cascade” quiere decir clasificador resultante en verdad está formado por clasificadores más simples (stages) que son aplicados sucesivamente en la ROI hasta que todos los “stages” pasan sin detectar nada. Los clasificadores básicos son clasificadores de tipo árbol con al menos dos hojas. Las entradas de los clasificadores básicos son las llamadas Haar-like features (feature= rasgo, característica), que se calculan con ayuda de la Figura 4.

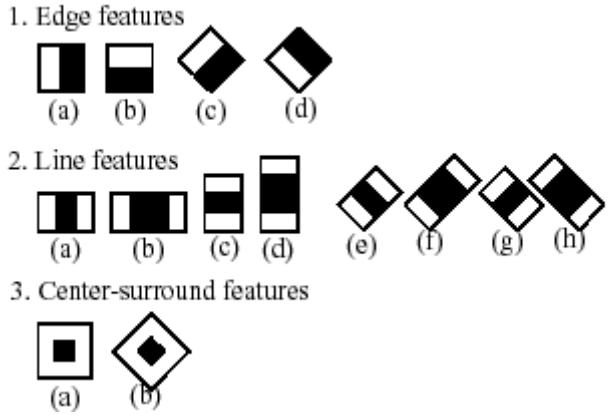


Figura 4: Haar-like features

El rasgo utilizado en un clasificador viene definido por su forma (1ª, 2b, etc.), posición en la ROI y la escala. Por ejemplo, en el caso del tercer rasgo de línea (2c) la respuesta se calcula como la diferencia entre la suma de los píxeles de la imagen en el rectángulo que abarca el rasgo entero (incluyendo las dos líneas blancas y la negra entre ellas) y la suma de los píxeles de la imagen por debajo de la raya negra, multiplicado por 3 con el fin de compensar las diferencias en el tamaño de las áreas. Las sumas de los valores de píxeles en regiones rectangulares se calculan rápidamente mediante el uso de la integral de imágenes.

3.HERRAMIENTAS Y PLATAFORMA

3.1 LIBRERIAS OPENCV

OpenCV ([5],[6],[14],[27],[28],[29]) es una librería desarrollada por Intel en 1999 cuyas siglas corresponden con los términos anglosajones Open source Computer Vision library. Es una librería creada para el tratamiento de imágenes, gracias a que proporciona un alto nivel de funciones para el procesado de imágenes está destinada principalmente a aplicaciones de visión por computador en tiempo real.

Está especialmente diseñada para la captura, tratamiento y visualización de imágenes en áreas como interfaz hombre-máquina, robótica, monitorización, etcétera. En la Figura 5 se muestra la estructura de la librería OpenCV.

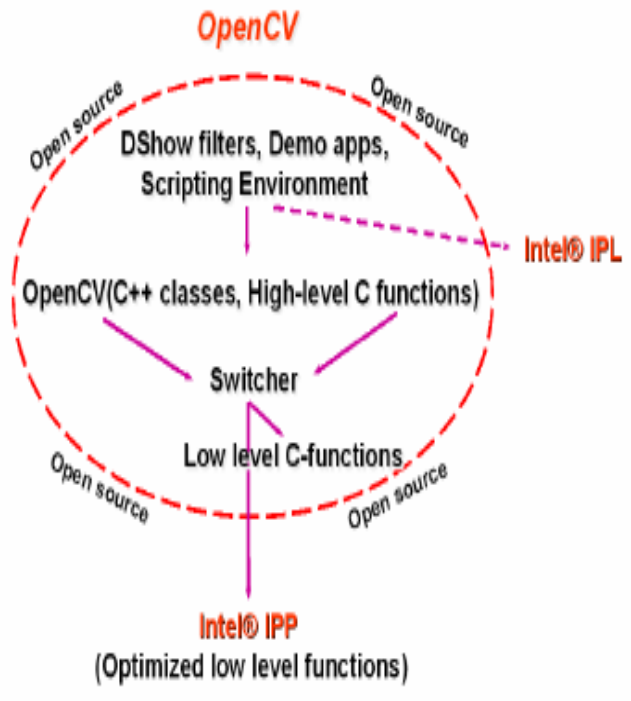


Figura 5: Estructura de la librería OpenCV

Conviene destacar que se trata de una librería libre, lo que permite su uso en los distintos Sistemas Operativos (Linux, Windows, Mac) sin reducir su funcionalidad. Desarrollada en lenguaje de programación C++ es compatible con IPL (Intel Processing

Library) y utiliza IPP (Intel Integrated Primitives), por lo que está optimizada para ser utilizada bajo procesadores Intel sin que ello excluya su uso bajo otro tipo de procesadores.

Hay interfaces disponibles para OpenCV con otros lenguajes y entornos:

- EiC: Intérprete ANSI C escrito por Ed Breen. No se actualiza desde el 2005 por lo que actualmente está en desuso.
- Ch: Intérprete ANSI C/C++ con características de scripting, desarrollado por SoftIntegration (www.softintegration.com). Los wrappers para Ch están disponibles en opencv/interfaces/ch.
- MATLAB: Entorno para procesamiento numérico y simbólico desarrollado por Mathworks. El interfaz MATLAB para algunas funciones de OpenCV se encuentra disponible en opencv/interfaces/matlab/toolbox. En cuanto a la integración con Matlab, OpenCV puede utilizar las estructuras nativas de Matlab, y es compatible con el Image Toolbox.
- Visual C++: En un proyecto de consola puede trabajar con las librerías OpenCV. Sólo hay que configurarlas en Visual C++.

Algunas de las posibilidades que permite OpenCV son:

- Operaciones básicas.
- Procesado de imágenes y análisis.
- Análisis de movimiento.
- Reconocimiento de objetos.
- Análisis de la forma y estructura de una imagen.
- Reconstrucción 3D y calibración de la cámara.
- Etcétera.

Aclaraciones

Hay tener especial cuidado en no confundir las funciones, con los tipos de datos propios de OpenCV. Para ello, la propia librería utiliza una sintaxis distintas para cada caso, con ligeras diferencias, por lo que si no se presta atención es fácil confundir ambas sintaxis.

Cada una de las funciones referenciadas en OpenCV comienza con las siglas “cv”, seguida del nombre de la función, con la primera letra de cada una de las palabras que componen dicho nombre en mayúscula. Por ejemplo: cvCreateImage, cvCloneImage, cvCvtColor, cvReleaseImage, etcétera.

En cambio para referirnos a los tipos de datos la sintaxis es muy similar a la de las funciones con la salvedad de que los tipos de datos empiezan con las siglas “Cv”. Por ejemplo: CvScalar, CvSeq, CvPoint, etcétera. Sin embargo hay tipos de datos que se declaran de forma distinta, como por ejemplo IplImage, que pasamos a explicar en profundidad ya que es la base de este proyecto.

```
Typedef struct _IplImage {

        int nSize;  /* tamaño de la estructura IplImage*/
```

```
int ID;  /* versión*/
int nChannels;  /* número de canales de la imagen */
int alphaChannel;  /* ignorado por OpenCV*/
int depth;  /* profundidad de la imagen en píxeles: 8, 16, con signo, …*/
char colorModel[4];  /* ignorado por OpenCV*/
char channelSeq[4];
int dataOrder;  /* permite separar o no los canales de una imagen*/
int origin;  /* origen de coordenadas de la imagen 0-arriba izq, 1-abajo dch*/
int align;  /* alineamiento de las filas de la imagen.  Ignorado */
int width;  /* anchura de la imagen en píxeles*/
int height;  /* altura de la imagen en píxeles*/
struct _IplROI *roi;  /*  puntero a la ROI si existe*/
struct _IplImage *maskROI;  /* nulo en OpenCV*/
void *imageId;  /* uso de la aplicación*/
struct  _IplTileInfo *tileInfo;  /* contiene información de teselado (repetición del mismo patrón)*/
int imageSize;  /* tamaño de la imagen en bytes*/
char *imageData;  /* puntero a la imagen alineada*/
int widthStep;  /* tamaño de alineamiento de columna en bytes*/
int BorderMode[4];  /* modos de los bordes*/
int BorderConst[4];  /* constantes para los bordes*/
char *imageDataOrigin;  /* punter a la imagen original*/
} IplImage;
```

Tipos de datos

OpenCV proporciona unos tipos de datos básicos para su utilización. También nos provee de tipos de datos introducidos como ayuda al programador, para hacer que el acceso a la información de interés sea más simple.

IplImage: como dijimos anteriormente se trata del tipo de datos básico de OpenCV. Con este tipo de datos se representan las imágenes sean del tipo que sean.

CvPoint y CvPoint2D32f: estos dos tipos de datos definen las coordenadas de un punto. CvPoint con números enteros y CvPoint2D32f con números decimales (punto flotante).

```
typedef struct CvPoint
{
int x;  /* coordenada x*/
int y;  /* coordenada y*/
}CvPoint;

typedef struct CvPoint2D32f
{
float x;  /* coordenada x*/
float y;  /* coordenada y*/
}CvPoint2D32f;
```

Estas estructuras suelen ir acompañadas de funciones que facilitan su uso y definición. En este caso son cvPoint y cvPoint2D32f.

Funciones importantes

En este apartado se explican las principales características de algunas de las funciones básicas del código.

- cvNamedWindow(char name, int type);

  Esta función me crea una ventana con el nombre introducido en la cadena de caracteres name. Si no pongo segundo parámetro o le pongo un 1, el tamaño de la ventana se ajustará automáticamente al de la imagen. Ejemplo:

  ```
  cvNamedWindow("ORIGINAL");
  ```

- cvShowImage(char name, CvArr *img);

  Esta función me muestra la imagen img en la ventana que tenga el nombre de la cadena guardada en name. Ejemplo:

  ```
  cvShowImage("ORIGINAL",final);
  ```

- cvDestroyAllWindows();

  Elimina todas las ventanas creadas mediante la función cvNamedWindow.

- cvReleaseImage(& CvArr *img);

  Esta función se encarga de liberar el espacio de memoria que ha sido asignado a una estructura CvArr*. El único parámetro que posee es el nombre de la imagen que se desea liberar. Ejemplo:

  ```
  cvReleaseImage( &final );
  ```

- cvLoadImage(char filename, int iscolor);

  Esta función me carga la imagen definida por el nombre introducido en filename, en la variable que le especifique. El parámetro iscolor me permite cargarla en el formato de color original o en blanco y negro. Ejemplo:

  ```
  IplImage* color = cvLoadImage("fotot.jpg", CV_LOAD_IMAGE_COLOR );
  ```

  Esta función me permite cargar imágenes desde diversos formatos: BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF y TIF.

Instalación

El proyecto se ha realizado en el Sistema Operativo Linux en su versión Ubuntu 10.04 LTS, también conocida como "Lucid Lynx". Este sistema operativo se ha instalado mediante un USB-LIVE descargado desde la página oficial de Ubuntu (www.**ubuntu**.com/desktop/get-**ubuntu**/download).

La instalación de la librería OpenCV se ha realizado compilándolo desde cero. Para ello tenemos que instalar previamente todo lo necesario para la compilación:

```
# sudo apt-get install build-essential cmake subversion libgtk2.0-dev pkg-config
```

A continuación instalamos los paquetes que me permiten operar con los distintos formatos de imagen:

```
# sudo apt-get install libpng12-0 libpng12-dev libjpeg62 libjpeg62-dev zlib1g zlib1g-dev libtiff4 libtiff4-dev libjasper1 libavcodec52 libdc1394-22
```

Es recomendable tener instalado Python y SWIG con sus respectivos dev. El siguiente paso es descargar el código fuente de OpenCV. Para ello se crea una carpeta, en mi caso creo la carpeta "codigo" en home. Abriendo el terminal y entrando:

```
# cd/home/código
```

Se ejecuta Subversion:

```
# svn co https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary/trunk
```

Una vez se ha terminado de descargar al directorio principal:

 # cd /home/código/trunk/opencv

A continuación se crea una carpeta y ejecutamos cmake:

 # mkdir release

 # cd release

 # cmake -D CMAKE_BUILD_TYPE = RELEASE -D CMAKE_INSTALL_PREFIX=/ usr/local -D BUILD_PYTHON_SUPPORT=ON ..

Para concluir con la instalación:

 # make

 # sudo make install

 # export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH

Por último si se desea comprobar que la instalación de las librerías se ha realizado correctamente:

 # cd /home/código/trunk/opencv/release/bin

 # ./cxcoretest

Para empezar a familiarizarse con las librerías OpenCv una buena idea es investigar los ejemplos y funciones ya compilados y ver cómo funcionan. Para ello:

 # cd /home/código/trunk/opencv/samples/c

 # sh ./build_all.sh

 # ./facedetect

Inconvenientes

En el caso del objeto de este proyecto (detección de objetos, cálculo de distancias,…) OpenCV no ofrece un producto completo sino algunas funciones a partir de las cuales puedes crear una base para obtener la funcionalidad deseada. Otro inconveniente es la necesidad de utilizar la librería IPL para tener acceso a funciones de bajo nivel. Sin embargo, estos inconvenientes no son realmente algo que merma la capacidad de OpenCV debido a la presencia de funciones muy interesantes.

Elección librerías OpenCV

Se han elegido las librerías OpenCV para el desarrollo de este proyecto ya que el uso de la visión por computador es grande y sigue aumentando. Al no existir mucha oferta en

interfaces de programación de aplicaciones, como OpenGL o DirectX para gráficos, la mayor parte del software de visión por computador se puede diferenciar en tres grupos:

- Código de investigación. En muchos casos lento, inestable o incompatible.
- Herramientas comerciales muy caras, como puede ser el caso de MATLAB+Simulink.
- Soluciones específicas acompañadas de hardware, por ejemplo sistemas de videovigilancia, controles de manufactura, etcétera.

Al tratarse de una librería estándar simplifica el desarrollo de nuevas aplicaciones y soluciones más sencillas.

Formato de las imágenes

Algunos de los formatos permitidos por OpenCV son BMP, TIFF, JPEG que se explican a continuación.

- Formato BMP

  Es un formato desarrollado por Microsoft para permitir una rápida entrada/salida por disco o pantalla.
  Una de sus características principales es la cantidad de niveles de profundidad que puede almacenar: 1 bit por píxel para imágenes de 2 colores (binarias), 4 bits por píxel para imágenes de 16 colores e imágenes de 24 bits por píxel o lo que es lo mismo 3 bytes para imágenes de color.
  Este formato utiliza compresión sin pérdida: RLE (Run-length encoding) o sin comprimir. Además, el almacenamiento es bottom-left (se empieza por la parte inferior izquierda), y entrelaza los canales.
  Ventajas:
  - No se pierde calidad de las imágenes.
  - Lectura y escritura rápidas.
  - Formato sencillo: cabecera y datos.

  Desventajas:
  - Tamaño de imágenes alto: ancho*alto*bits por píxel.
  - No adecuado para transmisión por red.
  - Poco popular fuera del entorno Microsoft Windows.

  Aplicaciones:
  - Aquellas que requieran rapidez en la salida por pantalla.
  - Aplicaciones en las cuales no se deba perder calidad de la imagen.

- Formato TIFF

  Creado por Aldus (ahora Adobe) y pensado para trabajos que requieran una impresión de alta resolución y calidad.
  Se trata de un formato muy flexible, basado en etiquetas que son bloques de datos de tamaño predefinido que contienen información sobre la imagen. Estas etiquetas se pueden crear y existen muchos tipo, una sola imagen puede tener varias.

Las características principales de este formato son: admite hasta 64.000 canales, un número arbitrario de bits por píxel (enteros o reales de 64 bits), distintos espacios de color, múltiples imágenes por fichero, cualquier tipo de compresión, etcétera. Esto le hace uno de los formatos más abiertos.
Ventajas:
- Independiente de la plataforma, flexible y ampliable.
- Se adapta a distintos tipos de necesidades.
- Puede contener ficheros con otros formatos.

Desventajas:
- Demasiado flexible por lo que es difícil crear programas que soporten todas sus características y tipos de etiquetas.
- Almacenamiento en tiras inadecuado según su uso.

Aplicaciones:
- Edición de fotografía de alta calidad.
- Impresión de carteles.
- Aplicaciones con necesidades especiales: imágenes multiespectrales, alta resolución de color,…

- Formato JPEG

Se trata de un formato bastante reciente, elaborado y orientado al almacenamiento de imágenes fotográficas.
Admite tanto imágenes en escala de grises como imágenes en color RGB. Además, incluye mecanismos avanzados de compresión, que pueden ajustarse a distintos ratios de compresión.
Una de sus principales características es la compresión con pérdida, mediante DCT (Discrete cosine transform). El fichero puede incluir una versión reducida para previsualizar la imagen antes de leerla.
Otro aspecto a tener en cuenta es que se encuentra libre de patentes.
Ventajas:
- En la mayoría de los casos consigue un ratio de compresión / calidad mucho mejor que en el resto de los formatos. Además, su nivel de compresión es ajustable entre 1:10 y 1:100.
- Se trata de un formato muy popular y casi exclusivo en muchos ámbitos.

Inconvenientes:
- Posee compresiones / descompresiones complejas y costosas.
- La información perdida no se recupera. Al trabajar con una imagen en JPEG guardándola en disco tras cada operación, la imagen se va degradando.

Aplicaciones:
- Se utiliza prácticamente en todas las aplicaciones de fotografía digital: captura, almacenamiento, transmisión, impresión,…
- No conviene utilizarlo si no se permite pérdida de calidad o si se trabaja con dibujos.

Tipos de imágenes

Las imágenes que se utilizan durante la implementación del código son de dos tipos, binarias y en color.

- Imágenes binarias.

  La principal característica de este tipo de imágenes es que sólo utilizan don niveles de intensidad para su codificación: el blanco y negro, o los bits 1 y 0.
  En las imágenes binarias a 1 píxel le corresponde 1 bit, por lo que se trata del tipo más sencillo que vamos a emplear.

- Imágenes en color.

  En las imágenes en color cada píxel se codifica con tres parámetros distintos: los colores en el caso de RGB y la tonalidad, saturación y brillo en el caso de HSV. Teniendo cada parámetro una profundidad de 1 byte.
  Por lo que cada pixel estará codificado con 3 bytes, uno por cada canal.

Además de la información detallada anteriormente cada píxel puede almacenar otro tipo de información en el dato depth de la estructura IplImage. Los distintos tipos son:

- IPL_DEPTH_8U: Enteros con signo de 8 bits (unsigned char).
- IPL_DEPTH_8S: Enteros con signo de 8 bits (signed char).
- IPL_DEPTH_16S: Enteros de 16 bits con signo (short int).
- IPL_DEPTH_32S: Enteros con signo de 32 bits (int).
- IPL_DEPTH_32F: Número en punto flotante de 32 bits (float).

3.2 HOAP3

La plataforma sobre la cual se desarrolla el presente proyecto es el humanoide HOAP-3, cuyas siglas corresponden al acrónimo inglés Humanoid for Open Architecture Platform, comercializado por la empresa japonesa Fujitsu. Se trata de la tercera versión fabricada por la marca tras el HOAP-1 (2001), HOAP-2 (2003).

Se podría definir como un robot humanoide "manejable", ya que a diferencia del resto de humanoides fabricados en su fecha, es un robot de reducidas dimensiones, con sólo 60 cm de altura y 8.8kg de peso. En la Figura 6 se muestra una foto del HOAP-3.

Figura 6: Robot humanoide HOAP-3

Consta de un procesador Pentium M de 1.1 GHz en su interior que funciona con el SO RT-Linux. La elección de este SO viene acompañado de un software de código abierto e información completa sobre la interfaz interna de hardware, lo que brinda al usuario de un control total para añadir funcionalidades. Incorpora dos cámaras, una en cada ojo, un altavoz, micrófono, LEDs, etcétera.

El hecho de tener unas dimensiones tan reducidas no implica una pérdida de movimiento ya que tiene 28 grados de libertad (GDL), distribuidos como se observa en la Figura 7.

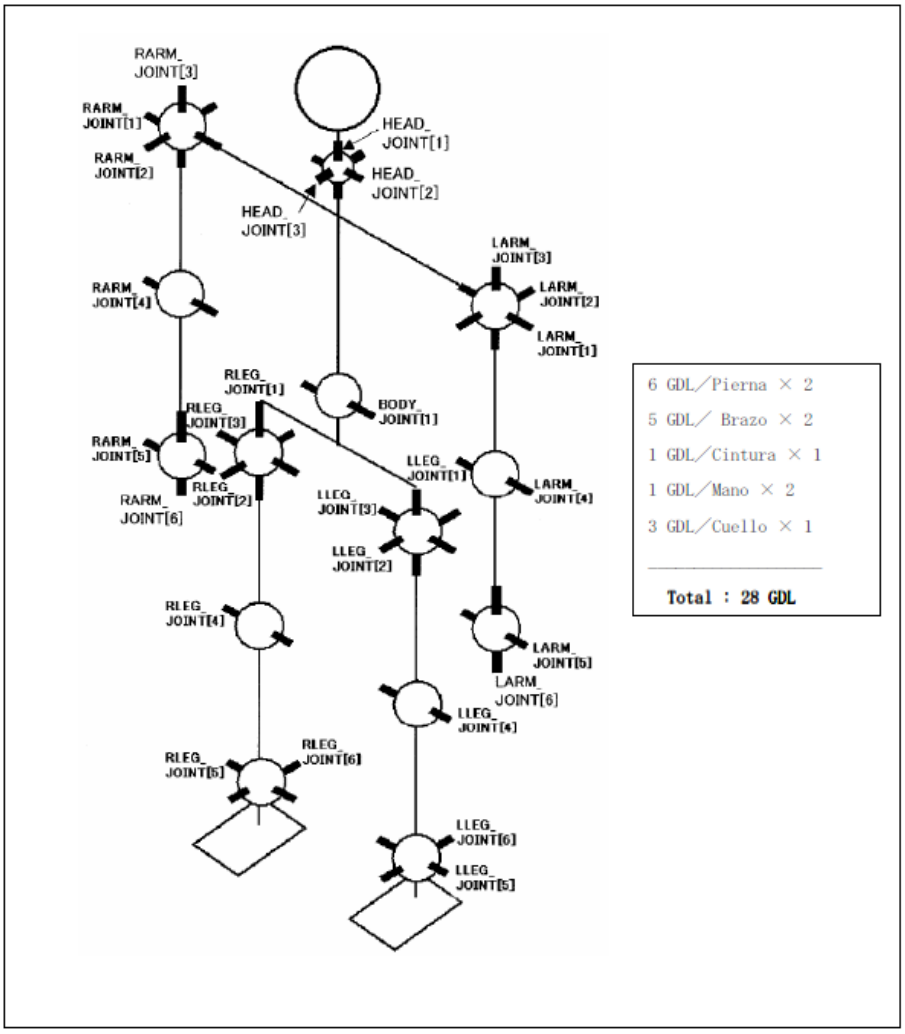


Figura 7: Esquema grados de libertad

A parte de los elementos mencionados anteriormente el robot HOAP-3 dispone también de sensores de distancia, de presión en las manos y los pies y de aceleración. Otro dispositivo que aumenta su funcionalidad es el hecho de incorporar conexión wifi, lo que permite tener una comunicación bidireccional entre el humanoide y un ordenador. La distribución de los dispositivos y sensores puede apreciarse en la Figura 8.

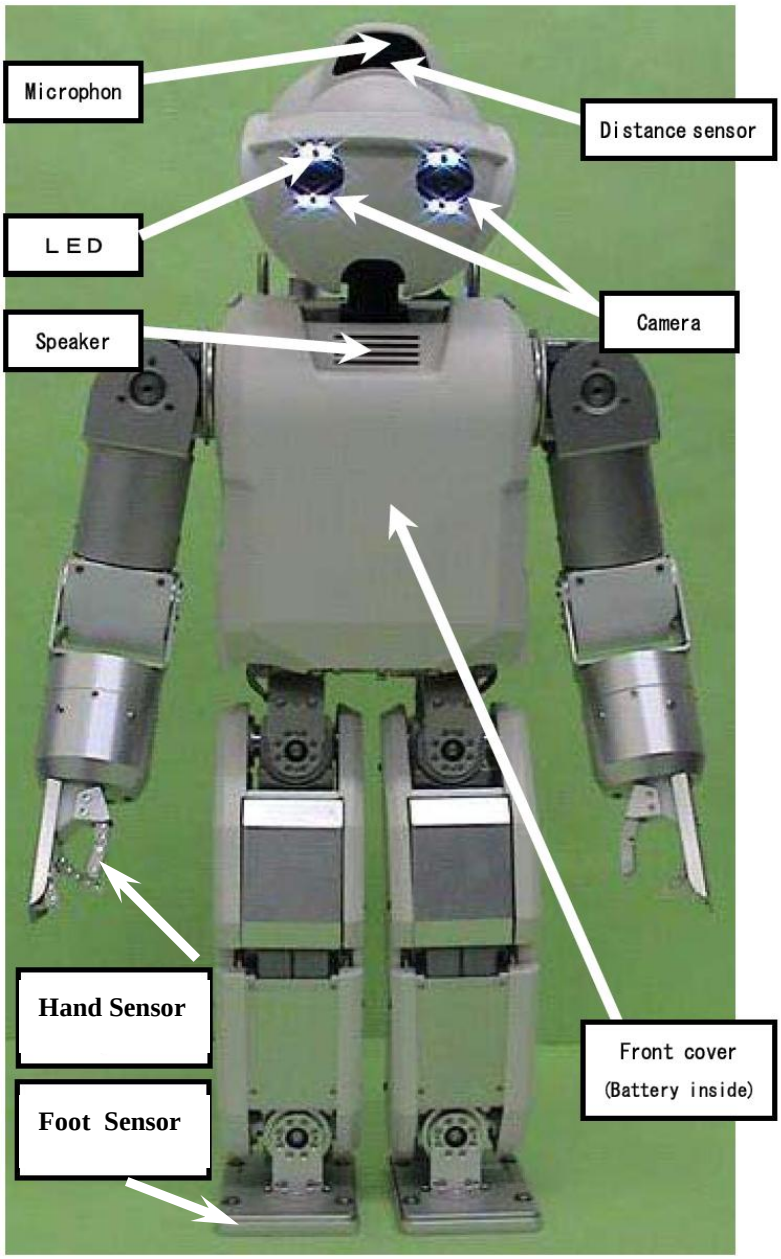


Figura 8: Posicionamiento de dispositivos

Entre las características que incluye figuran:

- La expresión de emociones mediante los LEDs que tiene alrededor de los ojos.
- Reconocimiento de imágenes y audio.

Síntesis de voz, con la consecuente posibilidad de comunicación con un humano.

El sistema de visión del robot humanoide se implemento con una arquitectura sencilla de cliente servidor. En la Figura 9: Figura 9 se muestra la arquitectura del sistema implementado.

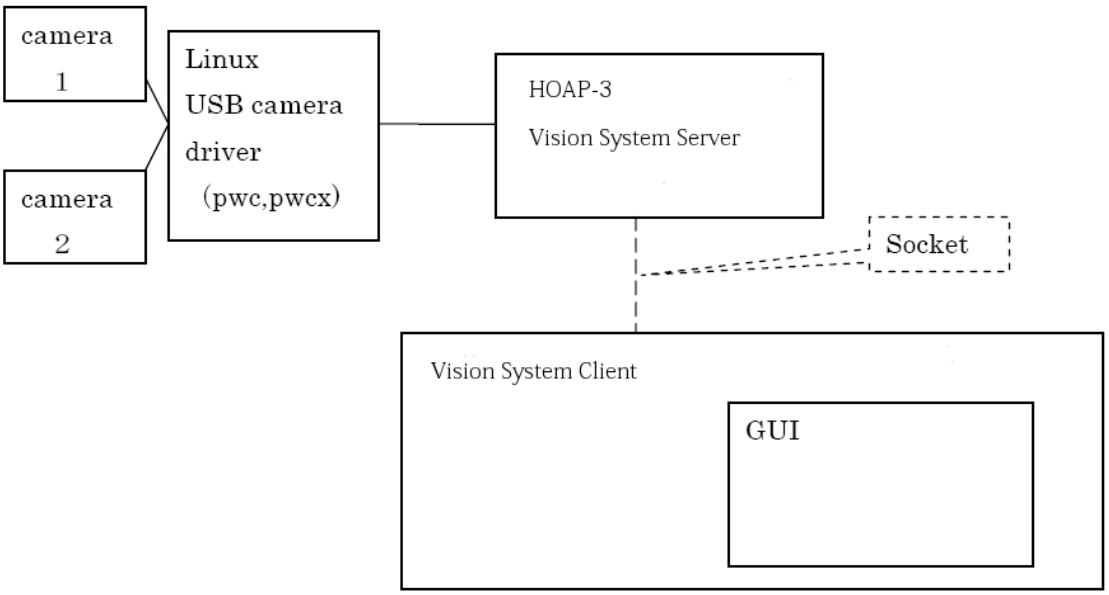


Figura 9: Arquitectura del sistema

El robot HOAP-3 cuenta con un sistema de dos cámaras que nos permite conseguir una visión estero. El servidor de visión, PC interno del HOAP-3, se encarga de capturar y envía la captura de imagen de las cámaras como un array de char. El cliente recibe la información de la imagen del robot en el formato YUV240p y luego convierte de este formato a la estructura IplImage requerida por algoritmos de detección e identificación descritos en la próxima sección.

La imagen procesada se muestra al usuario a través de una interfaz creada con las librerías QT. Las librerías QT y OpenCV se pueden integrar muy fácilmente y se puede cambiar de un formato de imagen a otro mediante las funciones QImage2IplImage e IplImage2QImage.

4.IMPLEMENTACION ALGORITMOS

Como bien se expuso en el capítulo 1 el objetivo de este proyecto es dotar al HOAP-3 de la capacidad de detectar objetos en su campo de visión y a partir de ahí que sea capaz de estimar la distancia a los mismos gracias a su visión estéreo. Para ello el algoritmo empleado se puede visualizar en el diagrama de flujo de la Figura 10. En la cual se pueden apreciar las distintas etapas por las que pasa para llevar a cabo la labor de detección y estimación de la distancia.

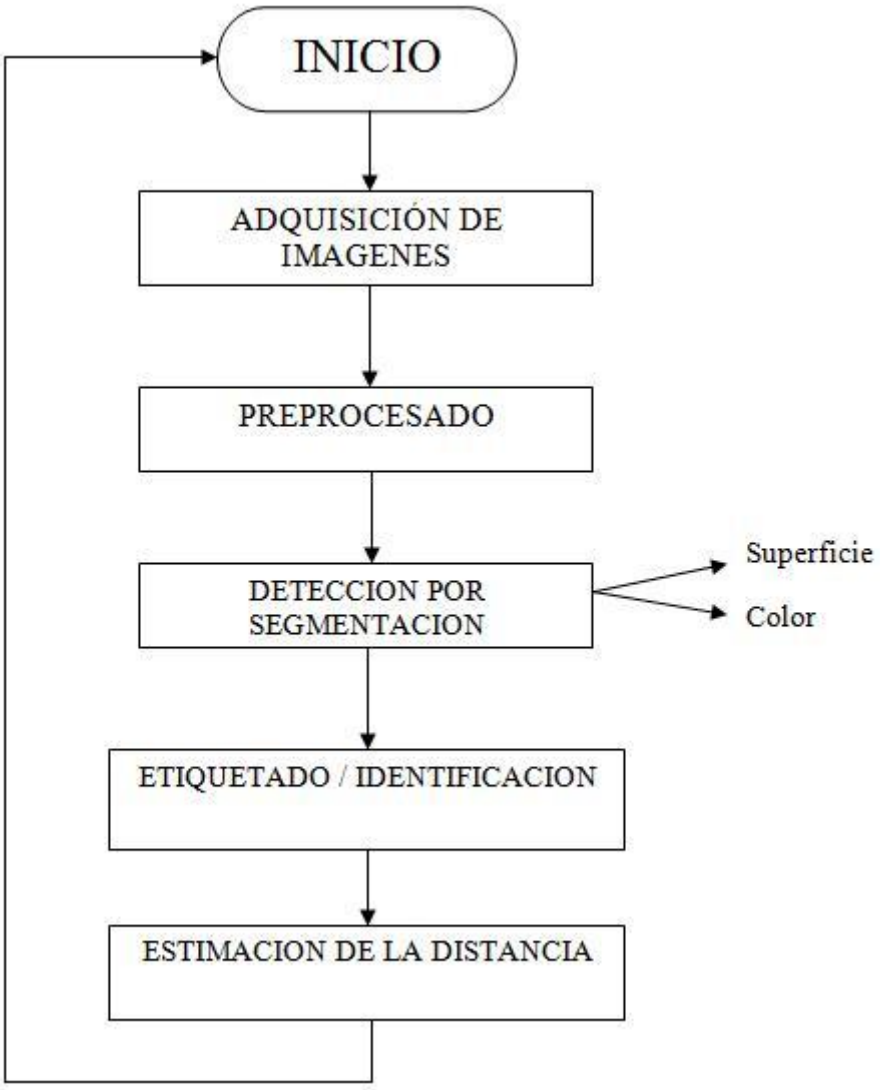


Figura 10: Diagrama de flujo del algoritmo principal

4.1 Detección de objetos

La detección de objetos es una aplicación muy importante dentro de la robótica, ya que confiere al humanoide de una funcionalidad añadida. Lo que le permitirá tomar decisiones en función de la información recibida.

Llevar a cabo una detección de objetos completa es muy complicado debido a la complejidad de la clase de objetos e imágenes. Para llevarla a cabo existen muchos métodos, muchos de los cuales se han explicado en el capítulo 2. La metodología empleada es la que se observa en los apartados, primero se procede con una serie de segmentaciones de la imagen para, a continuación encontrar bordes o contornos y así poder identificar el objeto que aparece en la imagen.

A continuación se exponen los algoritmos llevados a cabo para detectar uno o varios objetos.

4.1.1 Segmentación

La segmentación de la imagen es la primera de las operaciones / transformaciones que se llevan a cabo sobre la imagen recibida desde el humanoide. En visión por computador, los procesos de segmentación son de vital importancia cuando se trata de detectar objetos en entornos desestructurados. En el caso que nos ocupa, la segmentación haciendo uso de la información de color adquiere importancia en ámbitos muy diversos, desde el tratamiento de secuencias de vídeo para la detección de objetos o individuos en escenas móviles hasta la detección para realizar tracking (seguimiento), pasando por la detección de objetos para la manipulación mediante robots.

Para poder realizar la segmentación adecuada tendremos que definir bien las características que definen de manera unívoca a nuestro objeto u objetos de interés. A la que más importancia vamos a dar a priori va a ser al color, la otra característica diferenciadora de nuestro objeto u objetos será el área, que se comprobará en pasos posteriores.

Aquí surge una pregunta importante: ¿qué es color? La definición que propone la Real Academia Española (RAE) dice: "*Sensación producida por los rayos luminosos que impresionan los órganos visuales y que depende de la longitud de onda"*. Sin embargo no debemos olvidar que estamos trabajando con imágenes digitales y lo que nuestros ojos no perciben una cámara sí. Por esa misma razón trabajamos con el modelo HSV cuyas siglas provienen del acrónimo inglés Hue, Saturation, Value o lo que es lo mismo Tono, Saturación y Valor (Figura 11). Define un modelo de color en términos de sus componentes constituyentes en coordenadas cilíndricas:

- Tonalidad: tipo de color representado como un grado de un ángulo (0º-360º).
- Saturación: distancia al eje de brillo negro-blanco. También llamado pureza.
- Valor: brillo del color. Representa la altura en el eje blanco-negro.

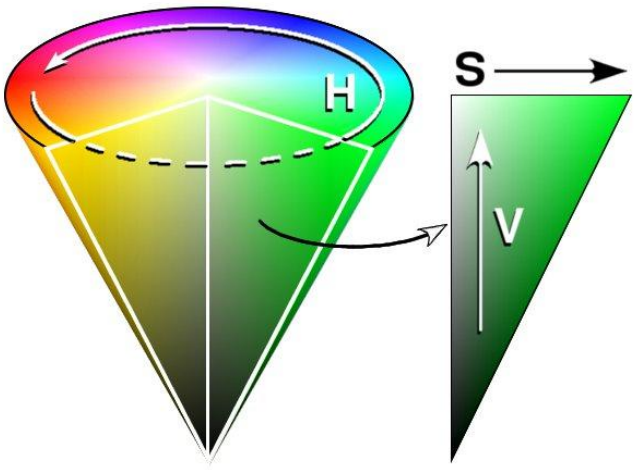


Figura 11: Representación espacial modelo de color HSV

El otro modelo de color más extendido es el modelo RGB del acrónimo inglés Red, Green y Blue (Figura 12). La desventaja de este modelo es que presenta menos información. Sin embargo se puede pasar de un modelo a otro aplicando simples fórmulas:

$$H_1 = cos^{-1}\left(\frac{\frac{1}{2}*((R-G)+(R-B)}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \qquad (3)$$

$$S = \frac{M-m}{M} \qquad (4)$$

$$V = \frac{M}{255} \qquad (5)$$

Siendo H = H_1 si $B \leq G$

$H = 360° - H_1,$ $si\ B > G$

M = max (R,G,B)

m = min(R,G,B)

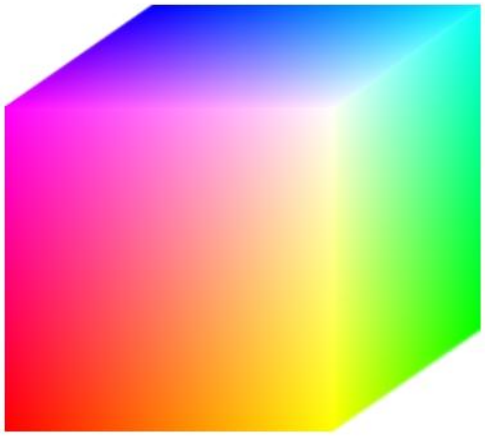


Figura 12: Representación espacial modelo de color RGB

Existen muchos más espacios de color en función de cómo modifiquen su representatividad, algunos a destacar son YUV, HSL, etcétera.

Una vez definida la característica principal el problema resulta en cómo distinguir el objeto deseado del resto del entorno. Para ello se utiliza la segmentación que entre otras funciones permite la extracción del fondo. Normalmente se han hecho dos aproximaciones básicas para resolver el problema de la segmentación. Una de ellas basada en la detección de bordes-contornos y otra basada en la detección de regiones.

En los métodos basados en bordes se detectan discontinuidades locales y que más tarde se tratan de unir formando un borde que delimita unos objetos de otros. Y en los métodos basados en regiones, se busca determinar las áreas de una imagen que tienen propiedades homogéneas y el borde de estas delimitará unos objetos de otros. Los dos métodos, por lo tanto, son complementarios y dependen en gran medida de la aplicación para la que vayan a ser destinados.

La segmentación empleada en este artículo se basa en la detección de regiones. Para poder distinguir las regiones se deben aplicar umbralizaciones. Tanto para RGB como para HSV la umbralización se basa en la detección de umbrales del mapa de componentes RGB o HSV. La umbralización de imágenes de objetos en entornos desestructurados muchas veces es compleja por la falta de conocimiento a priori del número de objetos a detectar o cambios en el entorno como por ejemplo la influencia de elementos no deseados como sombras, brillos, la complejidad de los colores, tamaño, posibilidad de solapamientos entre objetos, variaciones en el fondo, etcétera. Esto nos lleva a dos posibles problemas:

- Subsegmentación: elección de un número bajo de umbrales con lo que se obtienen menos regiones de las deseadas.
- Sobresegmentación: elección de un número alto por lo que se detectan más regiones de las deseadas.

Para evitar dichos contratiempos se ha decidido realizar una segmentación multinivel en el espacio HSV. La segmentación de una imagen HSV se realiza segmentando cada uno de las componentes de las que consta la imagen: tonalidad, saturación y valor para analizarlos y

trabajar con ellos. Otra razón para escoger el espacio HSV frente al RGB es porque puedo diferenciar de manera más precisa mi objeto de interés del resto de la imagen.

La detección de objetos es una utilidad muy importante en la robótica. Por ese motivo se ha intentado aumentar la funcionalidad del algoritmo mediante una función que nos haga posible diferenciar un objeto rectangular o cuadrado de uno circular. Esto permite que el humanoide no solo detecte el objeto sino que pueda coger uno u otro con criterio. Para lograr diferenciar la forma del objeto se hace una aproximación a partir del área y perímetro del objeto. Si podemos aproximar el área del objeto a la de un círculo con un margen de error de un 10% entonces el objeto es redondo, sino suponemos que el objeto es rectangular o cuadrado. La Figura 13 representa el diagrama de flujo para el proceso de segmentación.

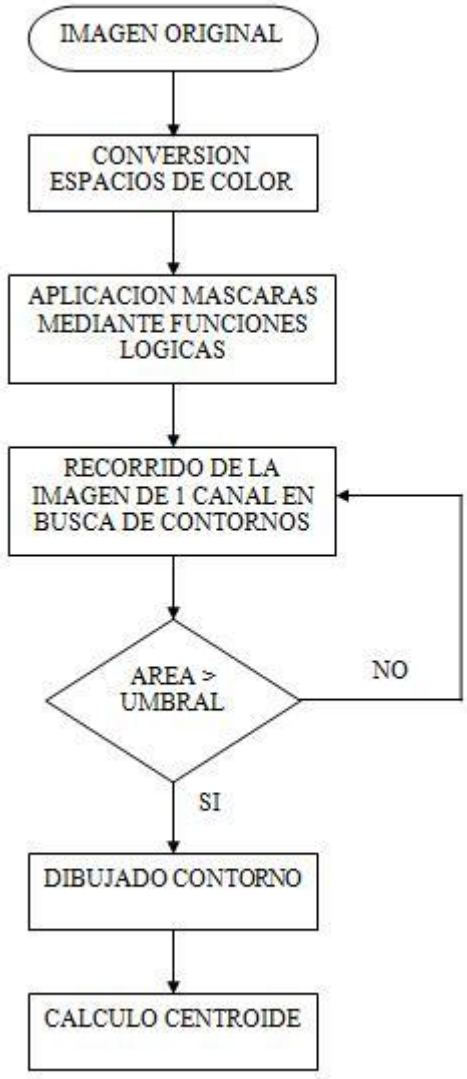


Figura 13: Diagrama de flujo de la segmentación

El proceso seguido en la segmentación ha sido el siguiente (Figura 13):

1. Transformo la imagen original de RGB a HSV.
   Esta operación se realiza de forma inmediata con una función de las librerías OpenCV: cvCvtColor(input,output,conversión_type). Sin embargo lo que se desea no es una sola imagen en el espacio HSV sino tres imágenes en cada una de las cuales tendré uno de los canales del modelo HSV, esto se consigue con la función cvSplit(input,output0,output1,output2,output3) siendo outpu3 cero en este caso.
2. Filtro la imagen original con la máscara de matiz.
3. Filtro la imagen original con la máscara de saturación.
4. Filtro la imagen original con la máscara de valor.

Como se ha explicado anteriormente para poder detectar un objeto es necesario realizar una segmentación, que en este caso se hará mediante segmentación por regiones. En los pasos 2-4 lo que se hace es crear unas máscaras, mediante umbralización basada en color, de cada uno de los canales del modelo HSV (Figura 14).

Figura 14: Máscaras en HSV

Cuando se trabaja en el sistema HSV, el color de cada uno de los píxeles de una imagen está definido por tres componentes: Tono, Saturación y Brillo o Valor. Para identificar los píxeles de un determinado color se comprueba que los niveles de sus tres componentes corresponden a los del color buscado. Por ello una umbralización por color tendrá la forma de:

$$g(x,y) = \begin{cases} 1 & si \begin{matrix} H_a \leq f_H(x,y) \leq H_b \\ S_a \leq f_S(x,y) \leq S_b \\ V_a \leq f_V(x,y) \leq V_b \end{matrix} \\ 0 & en\ cualquier\ caso \end{cases} \quad (6)$$

Donde $f_H(x,y)$, $f_S(x,y)$, $f_V(x,y)$ son, respectivamente, las funciones que dan los niveles de tono, saturación y brillo de cada uno de los píxeles de la imagen.

A partir de cada una de las máscaras obtenidas se aplican funciones lógicas. La función lógica empleada es una AND. OpenCV incluye una función para implementar esta operación: cvAnd(input1, input2, output). Al aplicar esta función lógica obtengo una imagen filtrada como puede observarse en la Figura 15.

Figura 15: Imagen original filtrada en HSV

5. Recorro la imagen obtenida en busca de contornos que cumplan las especificaciones.

4.1.2 Contornos

Una vez realizada la segmentación, y habiendo obtenido un resultado como el de la Figura 15, el siguiente paso sería proceder a la detección de contornos. En una imagen normal, es decir, sin segmentar, la cantidad de contornos que se pueden hallar es muy grande. Por este motivo se le pone restricciones de área. Si el área (en píxeles), interna del contorno es superior a un valor prefijado, entonces ese contorno es válido y por lo tanto se trata de uno de los objetos de interés. Si por el contrario es menor ese contorno se descarta por ser una lectura falsa.

4.1.3 Identificación

Cuando se ha determinado que en la imagen aparece un objeto de estudio, es el turno de identificarlo. El código implementado es capaz de diferenciar dos características fundamentales de cada objeto que se halle en la imagen:

- El color: esta característica la determina en la etapa de segmentación.
- La forma.

Las formas que puede detectar son pelota o taza. Para determinarla se realiza una suposición, si el área del contorno y por lo tanto del objeto, se aproxima con un margen de error del +/- 10% al área de un círculo entonces el objeto detectado es una pelota. Si no se aproxima a dicho área entonces se trata de la taza.

Para saber qué área debería tener teóricamente el círculo se calcula el perímetro del contorno. De ese perímetro se halla el radio del círculo teórico según la siguiente ecuación (7).

$$r = \frac{p}{2*\pi} \qquad (7)$$

Donde r: radio de la circunferencia.

p: perímetro del objeto / contorno detectado.

Cuando se ha calculado el radio teórico de la circunferencia se pasa a hallar el área del círculo que rodea con la ecuación (8). Si el área obtenido de la imagen cumple un margen de +/- 10% con respecto al área resultante de dicha ecuación entonces el objeto es una pelota.

$$A = \pi * r^2 \qquad (8)$$

Siendo r: radio de la circunferencia calculado anteriormente.

A: área de referencia. Calculado teóricamente.

4.2 Estimación de la distancia

El conocimiento de la distancia de un objeto al robot es muy importante. Este hecho va a permitir labores de tracking y grasping. Estas funcionalidades como se ha comentado en apartados anteriores son básicas para que junto con el resto de aplicaciones se consiga aumentar la versatilidad del robot.

La estimación de la distancia se puede realizar de dos formas distintas:

- Partiendo de una sola imagen.
- Con un par de imágenes.

4.2.1 Visión estéreo

Si partimos de imágenes monoculares hay que tener en cuenta aspectos como: iluminación, sombras, tamaño del objeto, distancia focal de la cámara, etcétera. Este método no es aconsejable ya que la distancia obtenida es una aproximación, lo que se traduce en falta de fiabilidad y precisión.

Como el humanoide está equipado con dos cámaras simulando ojos humanos, estas entradas pueden usarse para estimar la distancia usando el conocimiento acerca de la geometría de los sensores ópticos, es decir, la localización de las cámaras. En particular, cuando dos cámaras perciben un punto X de una misma escena, sus coordenadas espaciales pueden ser calculadas geométricamente usando la información de las imágenes.

Partiendo de un par de imágenes es mucho más sencillo y preciso. Esto se debe a que tenemos dos imágenes distintas del mismo entorno. Estas imágenes presentan una característica clave para el cálculo de distancias: la disparidad. La disparidad es la diferencia de la dirección horizontal en ambas imágenes. De una forma más intuitiva, la disparidad es la diferencia de posición de un punto fijo en ambas imágenes. Dicho punto puede ser el centroide de un objeto y la disparidad la distancia de dicho centro al margen izquierdo de las dos imágenes. A parte del centroide se deben conocer dos datos más:

- La distancia entre las cámaras (DIO: distancia intraocular).
- Los ángulos de visión horizontal y vertical de las cámaras.

La DIO suele variar entre 45-75 mm, en el caso del humanoide empleado se sitúa en 60mm. Cuanto mayor sea la DIO, mayor disparidad tendremos y por lo tanto, mejor se podrá calcular la distancia al objeto.

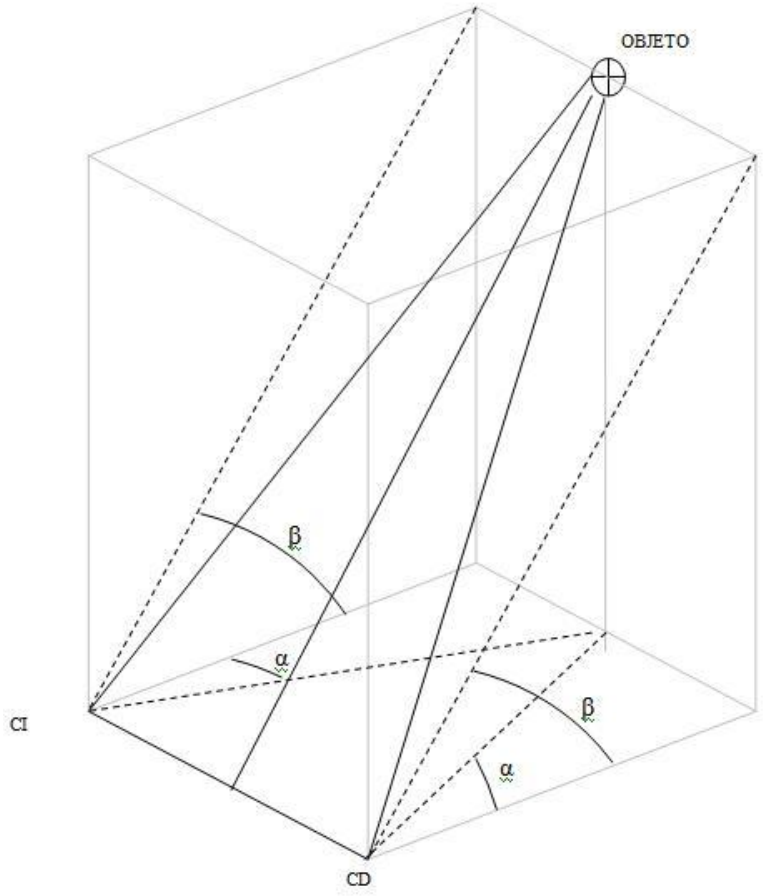


Figura 16: Sistema formado por las cámaras y el objeto

En la Figura 16 se puede observar un modelo simplificado del sistema formado por las cámaras del robot y el objeto. Basándose en dicho esquema se han aplicado las ecuaciones y fundamentos teóricos para el cálculo del objeto al centro de la línea formada por las cámaras (DIO).

4.2.2 Cálculos matemáticos

A continuación se explica el desarrollo matemático para el cálculo. Sólo se detallan los cálculos con una cámara pero hay que hacerlos para ambas:

$$\alpha = \left(\left(\frac{l}{2}\right) - xc1\right) * \left(\frac{H_{ANGLE}}{l}\right) \quad (9)$$

Donde: l: anchura de la imagen en píxeles.

xc1: posición x del centroide del objeto.

H_ANGLE: ángulo de visión horizontal de la cámara.

$$\beta = \left(\left(\frac{h}{2}\right) - yc1\right) * \left(\frac{V_{ANGLE}}{h}\right) \quad (10)$$

Donde: h: altura de la imagen en píxeles.

yc1: posición y del centroide del objeto.

V_ANGLE: ángulo de visión vertical de la cámara.

$$\frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)} \quad (11)$$

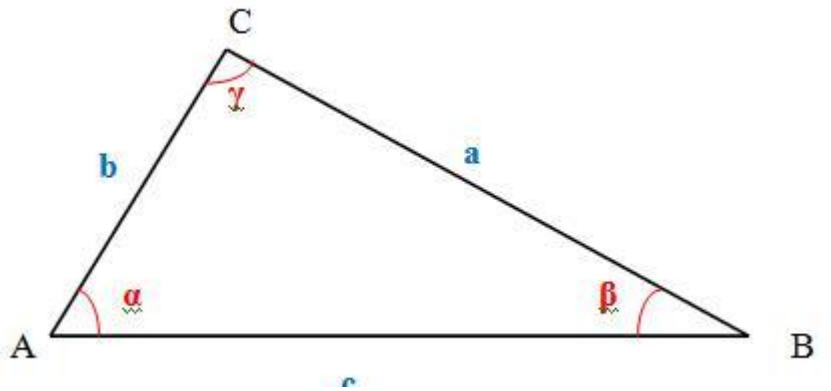


Figura 17: Representación genérica de un triángulo

El teorema del seno (11) establece que: la relación existente entre un lado dividido entre el ángulo opuesto se mantiene para los tres lados y sus respectivos ángulos (Figura 17).

Combinando las ecuaciones (9),(10),(11) y realizando proyecciones sobre los planos horizontal, vertical y lateral se obtiene la distancia desde el centro de la DIO al centro del objeto (Figura 18 y Figura 19).

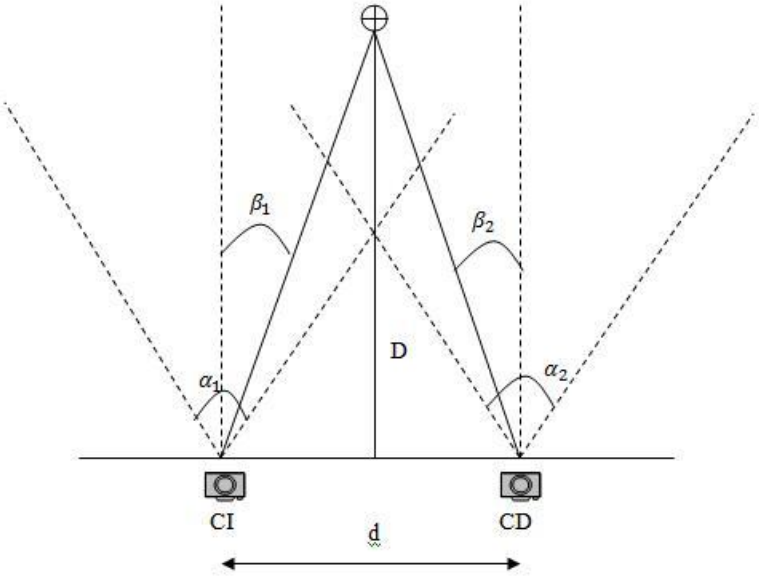


Figura 18: Representación horizontal del sistema cámaras-objeto

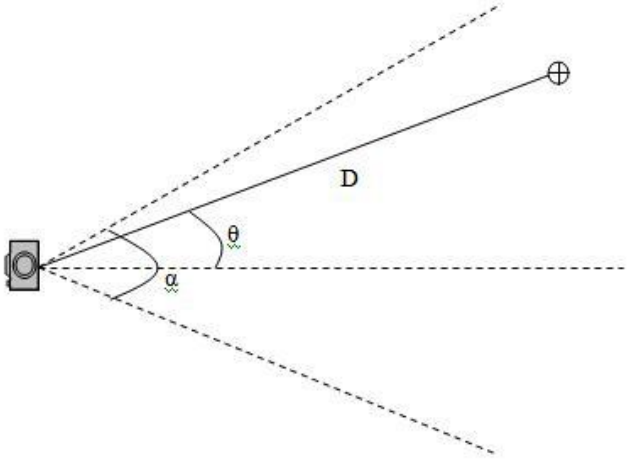


Figura 19: Representación lateral del sistema cámaras-objeto

4.3 Ajuste de parámetros y adaptación a diferentes objetos

El método empleado para la detección de objetos tiene ciertas condiciones:

- Por cada objeto que se quiera detectar hay que incluir una línea de código. Por objeto aquí se entiende color, ya que como se dijo anteriormente es la característica principal que define nuestros objetos de interés.
- Dependiendo de las condiciones del entorno, ya sea iluminación, fondo, etcétera, habrá que ajustar los parámetros de umbralización.

5.RESULTADOS, PROBLEMÁTICA Y LIMITACIONES

Durante el desarrollo del proyecto ha ido surgiendo problemas que han ayudado en la implementación de un algoritmo más robusto. Dichos contratiempos han sido de diversa índole. Desde problemas con el equipo utilizado hasta problemas en el desarrollo del código. A continuación se exponen los diferentes problemas y cómo se han ido resolviendo a lo largo de la ejecución del proyecto.

5.1 Software empleado

Uno de los primeros problemas que se han tenido que afrontar ha sido la instalación del software, tanto por parte del SO Linux, como por parte de las librerías OpenCV.

El problema con la instalación del SO ha sido que había problemas con el arranque con el USB-Live, ya que a veces arrancaba bien y otras no. Una vez instalado, las primeras veces que se iniciaba con Linux daba problemas. Sin embargo una vez actualizados los Kernels del SO no ha dado ningún problema que impida el correcto funcionamiento en Linux.

El otro gran problema ha sido la instalación completa de las librerías OpenCV. La instalación en Linux no es sencilla, pese a lo expuesto en el capítulo 3 la instalación no siempre se realiza con éxito. Esto se debe a que es probable que sea necesaria la instalación de librerías secundarias que me garantizan el funcionamiento de las librerías OpenCV.

5.2 Detección de objetos

En el capítulo 4 se ha explicado la importancia de usar bien los espacios de color sin embargo, ha sido un factor que no se tuvo en cuenta hasta la realización de las pruebas con el equipo externo, como comentamos en apartados siguientes. Otro factor importante a la hora de detectar el objeto

5.3 Estimación de la distancia

En las pruebas realizadas para comprobar la precisión del cálculo de la distancia no se ha observado ningún problema salvo en el caso comentado en el apartado 5.5

5.4 Pruebas en equipo externo

Por equipo externo entendemos el ordenador portátil dotado de webcam. Las pruebas realizadas con dicho material estaban supeditadas a varios aspectos en la metodología de trabajo:

- Análisis del resultado con una imagen.
- Análisis del resultado en vídeo.

En el primer caso la detección del objeto es muy simple ya que las condiciones del entorno, en este caso lo único a tener en cuenta es la iluminación, se mantienen estables permitiendo realizar un único ajuste de los parámetros para diferenciar los objetos. En la Figura 20 se observa el resultado obtenido de segmentar una imagen con varios objetos para extraer el fondo y a cada objeto asignarle un valor determinado de color.

Figura 20: Imagen segmentada según el modelo RGB

Como se puede observar en la Figura 20 se produce un fallo en la segmentación de la taza. El color que debería mostrar la taza es un azul como el que tiene el rotulador, nivel de azul asignado por el código. Otro inconveniente de trabajar en RGB se puede observar en la esquina superior derecha de la imagen o en la sombra de la taza, que me lo está detectando como azul.

En el segundo caso, el análisis del resultado en vídeo, uno de los problemas que han surgido ha sido el mismo que en apartados anteriores: la iluminación. Pese a estar trabajando ya con el espacio de color HSV esta variable sigue afectando al correcto funcionamiento del código. El problema que se puede observar en la **¡Error! No se encuentra el origen de la referencia.** es que pese a tener dos objetos del mismo color, si no se realiza una umbralización correcta el robot no es capaz de detectar ambos objetos del mismo color a la vez.

Figura 21: Problema de umbralización:

5.5 Pruebas en HOAP3

A parte de los problemas que se arrastran del apartado anterior, nos hemos enfrentado en esta etapa quizás al mayor problema hasta el momento. Dicho problema radica en que cada una de las cámaras obtiene una imagen ligeramente diferente de la otra cámara en cuanto a iluminación se refiere. Esto ha ocasionado que en ciertas ocasiones sólo detecte objetos una de las dos cámaras. Esto a su vez ha provocado que el robot no sea capaz de estimar la distancia al objeto, ya que como se ha comentado anteriormente para calcular una distancia de forma precisa es necesario contar con visión estéreo.

En la Figura 22 se muestra el caso de que sólo una de las cámaras es capaz de detectar el objeto.

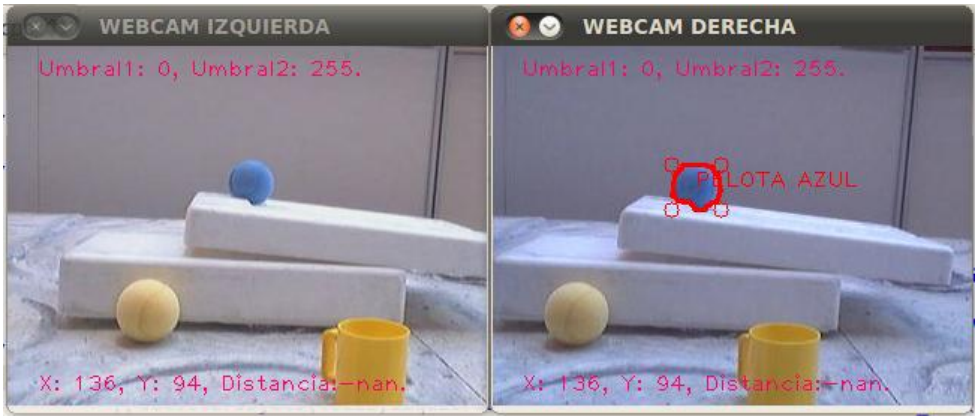


Figura 22: Problema con la calidad de imagen

Para subsanar este problema se ha tenido que realizar una umbralización independiente de cada una de las cámaras.

Una vez resuelto todos estos problemas surgidos durante el desarrollo del código, el resultado obtenido ha sido satisfactorio. Como se puede observar en la

Figura 23 la interfaz que nos muestra los objetos detectados también nos muestra varios aspectos:

- Número de objetos encontrados e identificación individualizada.
- Color y distinción de forma del objeto.
- Cálculo de las distancias de forma independiente.

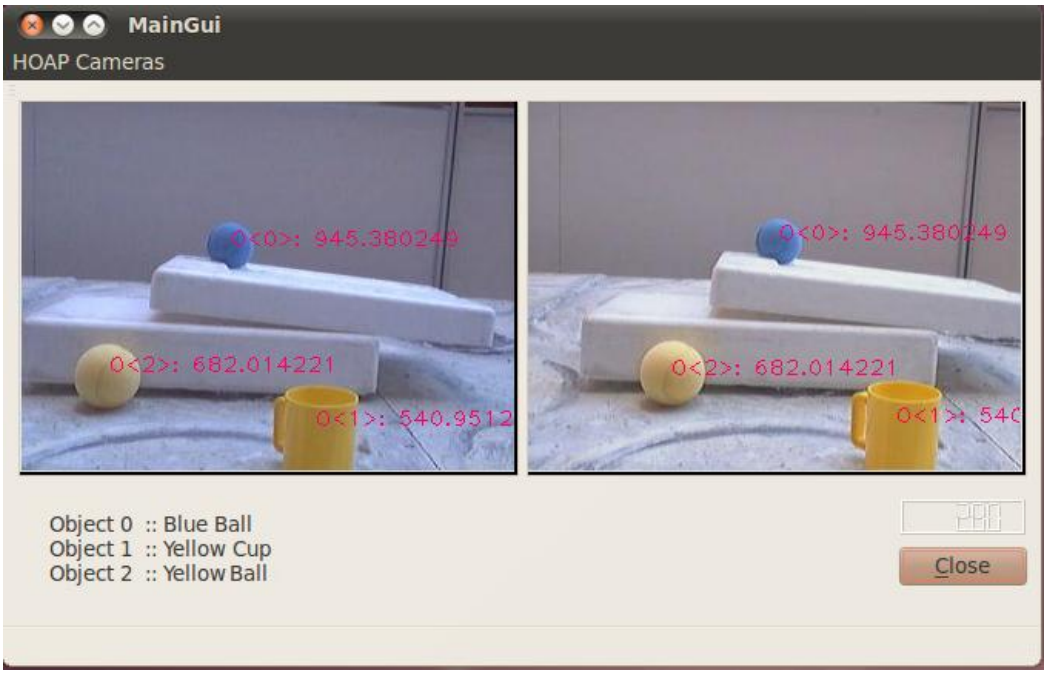
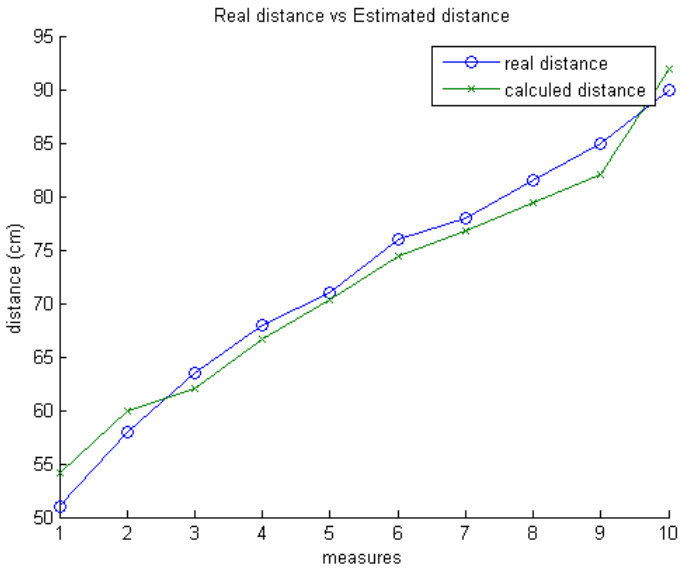


Figura 23: Resultado mostrado en la interfaz

Con las pruebas realizadas sobre la fiabilidad del algoritmo se ha llegado a los siguientes resultados: un error medio de 18mm en la estimación de la distancia, como puede verse en la Figura 24. El error más grande obtenido ha sido de un 6%.

Figura 24: Comparación distancia real y obtenida



6.CONCLUSIONES Y LINEAS DE TRABAJO FUTURO

El objetivo principal de este proyecto fin de carrera era el desarrollo de un código que permitiese al HOAP-3 detectar objetos y calcular las distancias a ellos para posteriormente ser capaz de realizar un tracking.

Para ello se ha realizado un estudio acerca de las principales metodologías para implementarlo. Como se ha detallado en esta memoria existen muchas formas de realizar esta tarea.

A partir de los resultados obtenidos en todas las pruebas desarrolladas durante su implementación se puede decir que el funcionamiento logrado es el deseado. Si bien es cierto que se podría haber realizado un algoritmo más robusto de haber dispuesto del material adecuado, ya que como se explica en el capítulo 3 la componente hardware del humanoide es básica.

Las líneas de trabajo futuro pueden orientarse de distintas maneras: continuar el desarrollo del algoritmo añadiéndole funcionalidades o intentar hacer más robusto el algoritmo de detección.

La primera línea consistiría básicamente en proceder a implementar funciones de tracking sobre los objetos detectados.

La segunda consistiría en tratar de implementar el algoritmo basado en Haar Feature-based Cascade Classifier para obtener un detector de objetos más robusto. Sin embargo habría que asegurarse previamente de que se dispone de la capacidad para permitir la ejecución de un algoritmo de esas características. Otra posible línea de trabajo futuro en esta misma dirección sería continuando con el algoritmo desarrollado e intentar hacerlo inmune a las variaciones externas de iluminación. Una forma posible sería simplificando la etapa de umbralización.

Los resultados obtenidos en este proyecto fin de carrera serán publicados en un artículo en las jornadas del séptimo Workshop de Robocity2030 (A. Peña et al., 2010) .

GLOSARIO

HOAP-3: Humanoid for Open Architecture Platform.
OpenCV: Open Source Computer Vision Library.
IA: Inteligencia Artificial.
ROI: Region Of Interest (región de interés).
IPL: Intel Processing Library.
IPP: Intel Integrated Primitives
RLE: Run-Length Encoding.
DCT: Discrete Cosine Transform.
RGB: Red, Green and Blue.
HSV: Hue, Saturation and Value.
SO: Sistema Operativo.
GDL: Grados de Libertad.
DIO: Distancia Intraocular.

BIBLIOGRAFIA

[1] D. Herrero-Pérez, P. Pierro, C. Balaguer. “Object Recognition and Guided-Grasping for Humanoid Robot in Unstructures Scenario”. Universidad Carlos III Madrid.

[2] Rainer Lienhart, Jochen Maydr. “An extended set of Haar-like features for rapid object detection. IEEE ICIP 2002.

[3] Paul Viola, Michael Jones. “Rapid Object Detection using a Boosted Cascade of Simple Features”. Accepted Conference on Computer Vision and Pattern Recognition 2001.

[4] Wan Najwa Binti Wan Ismail. “Object detection system using haar-classifier” Bachelor thesis. Mayo 2009.

[5] http://opencv.willowgarage.com/documentation/python/object_detection.html . Consultado Julio y Agosto 2010.

[6] Gary Bradski, Adrian Kaehler. “Learning OpenCV”. Ed. O´reilly. 2008.

[7] Dana H. Ballard, Christopher M. Brown. “Computer Vision”. Ed. Prentice-Hall. 1982.

[8] José A. Somolinos. “Avances en robótica y visión por computador”. Ediciones de la Universidad de Castilla la Mancha. 2002.

[9] Francisco Flórez. “Modelo de representación y procesamiento de movimiento para diseño de arquitecturas de tiempo real especializadas”. Tesis de doctorado. Universidad de Alcalá de Henares. 2001.

[10] Pablo Gil, Fernando Torres y Francisco G.Ortiz “Detección de objetos por segmentación multinivel combinada de espacios de color”. XXV Jornadas de Automática, Ciudad Real. 2004.

[11] Jorge Tarlea Jiménez “Sistema de posicionamiento de objetos mediante visión estéreo embarcable en vehículos inteligentes”.PFC, Universidad de Alcalá de Henares, Madrid. 2009

[12] Paul Zelanski, Mary Pat. Fisher “Color”. Editorial Tursen S.A. Edición Hermann Blume. 2001.

[13] Carlos Enrique Carleos “Manual programador final”. Universidad de Oviedo. 2005.

[14] http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5. Consultado Julio y Agosto 2010.

[15] Sara Pardo. “Aplicación del modelo BAG-OF-WORDS al reconocimiento de imágenes”. PFC. Universidad Carlos III Madrid. Julio 2009.

[16] Teddy V. Rojas, Wilmer Sanz, Francisco Arteaga. “Sistema de visión por computadora para la detección de objetos esféricos a través de la transformada de Hough”. Universidad de Carabobo. Venezuela. Abril 2008.

[17] Andrew Blower. "Development of a vision System for a Humanoid Robot". Bachelor Thesis. October 2001.

[18] Bruce G. Batchelor, Paul F. Whelan. "Intelligent Vision Systems for Industry". Springer-Verlaq. 1997.

[19] Daniele Nardi, Martin Riedmiller, Claude Sannut, José Santos-Victor. "RoboCup2004: Robot Soccer World Cup VIII". Springer-Verlaq. 2004.

[20] Tomás González. "Artificial Vision in the Nao Humanoid Robot". Máster Thesis. Universidad Rovira i Virgill. Septiembre 2009.

[21] Amaia Santiago. "Sistema de detección de objetos mediante cámaras. Aplicación en Espacios Inteligentes". PFC. Universidad de Alcalá. 2007.

[22] Ana González, Javier Martínez de Pisón, Alpha V. Pernía, Fernando Alba, Manuel Castejón, Joaquin Ordieres, Eliseo Vergara. "Técnicas y algoritmos básicos de visión artificial". Servicio de publicaciones de la Universidad de La Rioja. 2006.

[23] www.grupoalianzaempresarial.com/inteligenciaartificial.htm. Consultado Julio y Agosto 2010.

[24] http://www.webelectronica.com.ar/news21/nota09.htm. Consultado Julio y Agosto 2010.

[25] http://note.sonots.com/SciSoftware/haartraining.html. Consultado Julio y Agosto 2010.

[26] www.softintegration.com. Consultado Julio y Agosto 2010.

[27] http://gpiserver.dcom.upv.es/publications/pdf/object_tracking_2003.pdf. Consultado Julio y Agosto 2010.

[28] http://perception.inf.um.es/wiki/index.php/La_estructura_IplImage. Consultado Julio y Agosto 2010.

[29] http://sophia.javeriana.edu.co/~cbustaca/Vision_Artificial/presentaciones/pdf/Taller_OpenCV.pdf. Consultado Julio y Agosto 2010.

[30] V.S.Nalwa. "A Guided Tour to Computer Vision". Addison-Wesley. 1993

[31] Marshall y Martin 1993, Hamey y col. 1993

[32] http://acceda.ulpgc.es/bitstream/10553/2603/5/TutorialViolaJones.pdf. Consultado Julio y Agosto 2010.

ANEXOS

A.1 Código desarrollado

```
/*********************************************************************
*     ARCHIVO:    detec_dist.cpp
*     AUTOR:      Alberto Peña Baeza
*     FUNCION:    Objects detection and distances calcutation
*********************************************************************/
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

#include "camparam.h"
#include "common.h"
#include <math.h>


IplImage* segmentarObjeto(IplImage* input,IplImage* output, int
umbral1, int umbral2, int canal )   //Segmentation of the image
obtaining the masks.
{
      IplImage*  imagen = cvCloneImage(input);
      IplImage* mascara1 = cvCreateImage( cvGetSize(imagen), 8, 1 );
      IplImage* mascara2 = cvCreateImage( cvGetSize(imagen), 8, 1 );
      IplImage* mascara = cvCreateImage( cvGetSize(imagen), 8, 1 );
      IplImage* canales[3] = {0,0,0};
      for( int i=0; i<3; i++ )
            canales[i]=cvCreateImage( cvGetSize(imagen), 8, 1 );

      cvSplit(imagen,canales[0],canales[1],canales[2],0);
      //3 channel image split in 3 images of 1 channel
      cvThreshold(canales[canal],mascara1,umbral1,255,CV_THRESH_BINARY
);    //apply a threshold to the image
      cvThreshold(canales[canal],mascara2,umbral2,255,CV_THRESH_BINARY
_INV );
      cvAnd(mascara1,mascara2,mascara);
      cvCvtColor(mascara,output,CV_GRAY2BGR);

      cvReleaseImage(&imagen);
      cvReleaseImage(&mascara1);
      cvReleaseImage(&mascara2);
      cvReleaseImage(&mascara);
      for( int i=0; i<3; i++ )
            if(canales[i])
                  cvReleaseImage(&canales[i]);
      return output;
}

IplImage* contornos(IplImage* input, IplImage* output, char cad[50],
CvMemStorage* storage, CvScalar colorTexto, int *dircx, int *dircy)
{
      IplImage*  imagen = cvCloneImage(input);
      IplImage* img_8uc1 = cvCreateImage( cvSize(imagen->width,imagen-
>height), 8, 1 );
      IplImage* img_edge = cvCreateImage( cvGetSize(img_8uc1), 8, 1 );
```

```
	CvPoint2D32f pt[4];
	CvPoint p[4];
	CvBox2D caja;

	CvSeq *first_contour = NULL,*result;

	cvCvtColor(imagen,img_8uc1,CV_RGB2GRAY);
	cvThreshold( img_8uc1, img_edge, 80, 255, CV_THRESH_BINARY );


	float r;
	int Nc = cvFindContours(img_edge,storage,&first_contour,
sizeof(CvContour),CV_RETR_EXTERNAL,CV_CHAIN_APPROX_SIMPLE );
	int n=0;
	for( CvSeq* c=first_contour; c!=NULL; c=c->h_next )
	//Find all the contours but only draw the good ones.
	{
		result = cvApproxPoly(c, sizeof(CvContour), storage,
CV_POLY_APPROX_DP, cvContourPerimeter(c)*0.003, 0 );

		if(fabs(cvContourArea(result,CV_WHOLE_SEQ))>300
	//Minimun possible area of the object
		&& fabs(cvContourArea(result,CV_WHOLE_SEQ))<100000)
	//Maximun possible area of the object
		{

	cvDrawContours(output,result,CV_RGB(255,0,0),CV_RGB(0,0,255),0,2
,8);
			cvBoxPoints(caja,pt);
			CvPoint centro = cvPoint(0,0);

			for( int i = 0; i < 4; i++ )
			{
				p[i] = cvPointFrom32f( pt[i] );
				centro.x+=p[i].x;
				centro.y+=p[i].y;
				cvCircle(output,p[i],5,colorTexto,1);
			}

			centro.x = centro.x/4.0;
			centro.y = centro.y/4.0;

////////////////////////////////////////////////////////////////
//CALCULATE THE PERIMETER TO DISTINGUISH BALL OR SQUARE OBJECT
////////////////////////////////////////////////////////////////
			int perimeter=cvContourPerimeter(c);
	//cvArcLength(first_contour); Another way to obtain perimeter.
			printf("El perimetro es:%d\n", perimeter);

			r = perimeter/(2*PI);
			int area = fabs(cvContourArea(c,CV_WHOLE_SEQ));
			int objeto; //If 0 is a ball, if 1 is a cup
			if (area = PI*r*r)	//(area>(0.90*PI*r*r)) ||
(area<(1.1*PI*r*r)))
				objeto = 0;
			else
```

```
                        objeto = 1;
                    printf("el objeto es:%d\n",objeto);
                    cvPutText(output,cad,centro,&cvFont(1.0),colorTexto);

                    *dircx=centro.x;
                    *dircy=centro.y;

              }
        }

        cvReleaseImage(&imagen);
        cvReleaseImage(&img_8uc1);
        cvReleaseImage(&img_edge);
        return output;
}

IplImage* detectarObjeto(IplImage* input, IplImage* output, int
color1, int color2, int sat1, int sat2, int brillo1, int brillo2,
                                                    char cad[50],
CvScalar color, CvMemStorage* storage, int *centrox, int *centroy)
{
        int dircx,dircy;
        dircx=*centrox;
        dircy=*centroy;
        IplImage* hsv = cvCreateImage( cvGetSize(input), 8, 3 );
        IplImage* mascara = cvCreateImage( cvGetSize(input), 8, 3 );
        IplImage* mostrar = cvCreateImage( cvGetSize(input), 8, 3 );
        ////////////////////////////////////////////////////////////////
        cvCvtColor(input,hsv,CV_BGR2HSV);         // Change from BGR to
HSV.
        segmentarObjeto(hsv,mascara,color1,color2,0);         // Hue
mask.
        cvAnd(input,mascara,mostrar);             // Filter between the
original image and the Hue mask.

        cvCvtColor(mostrar,hsv,CV_BGR2HSV);       // Change from BGR to
HSV.
        segmentarObjeto(hsv,mascara,sat1,sat2,1);       // Saturation
mask.
        cvAnd(input,mascara,mostrar);             // Filter between the
original image and the Saturation mask.

        cvCvtColor(mostrar, hsv, CV_BGR2HSV);           // Change from
BGR to HSV.
        segmentarObjeto(hsv,mascara,brillo1,brillo2,2);       // Value
mask.
        cvAnd(input,mascara,mostrar);             // Filter between the
original image and the Value mask.

        contornos(mascara,output,cad,storage,color, &dircx, &dircy);
        // Draw contour of the object.
        ////////////////////////////////////////////////////////////////
////////////////////////////////////
        *centrox=dircx;
        *centroy=dircy;

        cvReleaseImage( &hsv );
        cvReleaseImage( &mostrar );
        cvReleaseImage( &mascara );
```

```
}


int distanciaObjeto(IplImage *todiz, IplImage *tode, int xc1, int xc2,
int yc1, int yc2, int horiz, int vert, float *distancia)
{
      float base_line = 60;
      float focal_lenght = 55;


      float azi1 = ((todiz->width/2) - xc1) * (H_ANGLE/todiz->width);
      float azi2 = ((tode->width/2) - xc2) * (H_ANGLE/tode->width);

      float ele1 = ((todiz->height/2) - yc1) * (V_ANGLE/todiz-
>height);
      float ele2 = ((tode->height/2) - yc2) * (V_ANGLE/tode->height);


      //DISTANCE FROM THE CAM PROYECTED IN THE HORIZONTAL PLANE WITH
SINE'S THEOREM
      float tsen1 = base_line / sin(azi1-azi2);
      float dih = tsen1 * sin(RAD(90)-azi2);
      float ddh = tsen1 * sin(RAD(90)-azi1);

      //DISTANCE IN THE LATERAL PLANE WITH SINE'S THEOREM
      float tsen2 = ddh / sin(RAD(90));
      float tsen3 = dih / sin(RAD(90));
      float ddl = tsen2 * sin(RAD(90) - azi2);
      float dil = tsen3 * sin(RAD(90) - azi1);

      //LATERAL DISTANCE
      float tsen4 = ddl / sin(RAD(90) - ele2);
      float tsen5 = dil / sin(RAD(90) - ele1);
      float ddv = tsen4 * sin(RAD(90));
      float div = tsen5 * sin(RAD(90));

      //REAL DISTANCE WITH SINE'S THEOREM
      float tsen6 = ddv / sin(RAD(90) - azi2);
      float tsen7 = div / sin(RAD(90) - azi1);
      float dd = tsen6 * sin(RAD(90));
      float di = tsen7 * sin(RAD(90));


      //TOTAL DISTANCE FROM THE CENTER BETWEEN CAMS
        //Cosine's Theorem to calculate angles
      float alpha = acos(((dd*dd)-(di*di)-(base_line*base_line))/(-
2*di*base_line));
      float beta = acos(((di*di)-(dd*dd)-(base_line*base_line))/(-
2*dd*base_line));
      float gamma = acos(((base_line*base_line)-(dd*dd)-(di*di))/(-
2*dd*di));

      //Sine's Theorem to calculate total distance
      float a = RAD(180) - beta - (gamma/2);
      float b = RAD(180) - alpha - (gamma/2);
      //float r = dd / sin(a);
      float m = di / sin(b);
```

```
	//float d1 = r * sin(beta);
	float d2 = m * sin(alpha);


	if (d2<0)
		d2=-d2;

	printf("La distancia al objeto es d=%f\n",d2);

	*distancia=d2;



}


// Main function
int main(int argc, char* argv[])
{

	IplImage* todiz = cvLoadImage("todiz.jpeg", CV_LOAD_IMAGE_COLOR
);
	IplImage* tode = cvLoadImage("tode.jpeg", CV_LOAD_IMAGE_COLOR );

	//CvCapture* capture = cvCaptureFromCAM(0);	//To get the
image from webcam
	CvMemStorage* storage = cvCreateMemStorage();

	//IplImage* color = cvQueryFrame( capture );	//If we use image
from webcam
	IplImage* izquierda= cvCreateImage( cvGetSize(todiz), 8, 3 );
	IplImage* derecha= cvCreateImage( cvGetSize(tode), 8, 3 );

	CvFont fuente = cvFont(1.0);


	int centrox;
	int centroy;

	int xc1, xc2, yc1, yc2;		//X and Y coordinates of the
mass centers of the object from each webcam
	float distancia;

	int umbral1 = 0;
	int umbral2 = 255;

		cvNamedWindow("WEBCAM IZQUIERDA");
	cvNamedWindow("WEBCAM DERECHA");
	cvMoveWindow("WEBCAM IZQUIERDA", 0, 0);
	cvMoveWindow("WEBCAM DERECHA", 500, 0);


	while(1)
	{

////////////////////////////////////////////////////////////////////
//LEFT WEBCAM
////////////////////////////////////////////////////////////////////
		//color = cvQueryFrame( capture );
		izquierda = cvCloneImage(todiz);	// Image cloned
```

```
            detectarObjeto(todiz,izquierda,100,120,115,150,0,255,"TAZA
AZUL",CV_RGB(255,0,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);

      detectarObjeto(todiz,izquierda,100,120,75,145,115,225,"PELOTA
AZUL",CV_RGB(255,0,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(todiz,izquierda,50,80,125,255,150,255,"CAJA
VERDE",CV_RGB(0,255,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(todiz,izquierda,0,180,30,255,0,50,"ESTUCHE
NEGRO",CV_RGB(0,0,255),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(todiz,izquierda,0,50,110,175,80,255,"PELOTA
AMARILLA",CV_RGB(255,255,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(todiz,izquierda,0,30,95,255,0,255,"TAZA
AMARILLA",CV_RGB(255,255,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);

            xc1=centrox;
            yc1=centroy;

            char cad[50], cad2[50];
            sprintf(cad,"Umbral1: %d, Umbral2: %d.",umbral1,  umbral2);
            sprintf(cad2,"X: %d, Y: %d, Distancia:%f.",centrox,
centroy, distancia);
            cvPutText( izquierda, cad,
cvPointFrom32f(cvPoint2D32f(20,20)), &fuente, CV_RGB(255,0,100) );
            cvPutText( izquierda, cad2,
cvPointFrom32f(cvPoint2D32f(20,230)), &fuente, CV_RGB(255,0,100) );
            cvShowImage("WEBCAM IZQUIERDA",izquierda);

            printf("El centro de mi objeto en la cam izqda es:%d,%d\n",
xc1, yc1);

////////////////////////////////////////////////////////////////////
//RIGHT WEBCAM
////////////////////////////////////////////////////////////////////
            //color = cvQueryFrame( capture );
            derecha = cvCloneImage(tode); // Clono la imagen original.

            detectarObjeto(tode,derecha,100,120,115,150,0,255,"TAZA
AZUL",CV_RGB(255,0,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(tode,derecha,100,120,75,145,115,225,"PELOTA
AZUL",CV_RGB(255,0,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(tode,derecha,50,80,125,255,150,255,"CAJA
VERDE",CV_RGB(0,255,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(tode,derecha,0,180,30,255,0,50,"ESTUCHE
NEGRO",CV_RGB(0,0,255),storage,&centrox,&centroy);
            cvClearMemStorage(storage);
            detectarObjeto(tode,derecha,0,50,110,175,80,255,"PELOTA
AMARILLA",CV_RGB(255,255,0),storage,&centrox,&centroy);
```

```
            cvClearMemStorage(storage);
            detectarObjeto(tode,derecha,0,30,95,255,0,255,"TAZA
AMARILLA",CV_RGB(255,255,0),storage,&centrox,&centroy);
            cvClearMemStorage(storage);

            xc2=centrox;
            yc2=centroy;

            //char cad[50];
            sprintf(cad,"Umbral1: %d, Umbral2: %d.",umbral1, umbral2);
            sprintf(cad2,"X: %d, Y: %d, Distancia:%f.",centrox,
centroy, distancia);
            cvPutText( derecha, cad,
cvPointFrom32f(cvPoint2D32f(20,20)), &fuente, CV_RGB(255,0,100) );
            cvPutText( derecha, cad2,
cvPointFrom32f(cvPoint2D32f(20,230)), &fuente, CV_RGB(255,0,100) );
            cvShowImage("WEBCAM DERECHA",derecha);

            printf("El centro de mi objeto en la cam dcha es:%d,%d\n",
xc2, yc2);


      distanciaObjeto(todiz,tode,xc1,xc2,yc1,yc2,H_ANGLE,V_ANGLE,&dist
ancia);


////////////////////////////////////////////////////////////////////
//TO ADJUST HUE, SATURATION AND VALUE THRESHOLDS
////////////////////////////////////////////////////////////////////
            char c = cvWaitKey(33);
            if(c==27)
                  break;
            if(c=='+')
            {
                  umbral1+=5;
                  if( umbral1 > 255 )umbral1=0;
            }
            if(c=='-')
            {
                  umbral1-=5;
                  if( umbral1 < 0 )umbral1=255;
            }
            if(c=='*')
            {
                  umbral2+=5;
                  if( umbral2 > 255 )umbral2=0;
            }
            if(c=='/')
            {
                  umbral2-=5;
                  if( umbral2 < 0 )umbral2=255;
            }

      }


      cvDestroyAllWindows();
      //cvReleaseCapture( &capture );
      cvReleaseImage( &derecha );
      cvReleaseImage( &izquierda );
```

```
        cvReleaseMemStorage( &storage );
        return 0;

}
```